НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ Ролік О.І.
(підпис) (ініціали, прізвище)
“___”_____2019

Дипломний проект

освітньо-кваліфікаційного рівня «бакалавр»

зі спеціальності 6.050201 «Системна інженерія»
(код і назва)

на тему: Методи розподілу ресурсів критичної IT-інфраструктури

Виконав: Субботіна Лідія Сергіївна _____
(підпис)

Керівник: Дорогий Я.Ю. _____
(підпис)

Рецензент доцент кафедри ТК, к.т.н. Ткач М. М. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах
Ступінь вищої освіти перший (бакалаврський)
Спеціальність 6.050201 «Системна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____О.І.Ролік
(підпис) (ініціали, прізвище)
“__”_____2019 р.

ЗАВДАННЯ

на дипломний проект освітньо-кваліфікаційного рівня "бакалавр"

Студенту Субботіній Лідії Сергіївні
(прізвище, ім’я, по батькові)

1. Тема проекту Методи розподілу ресурсів критичної ІТ-інфраструктури
керівник проекту Дорогий Ярослав Юрійович, к.т.н., доц. ,

затверджені наказом по університету від «__» _____ 2019 р. №_____

2. Термін подання студентом проекту «__» _____ 2019 р.

3. Вихідні дані до проекту _____
_____
_____

4. Перелік питань, які мають бути розроблені:

_____
_____
_____
_____

5. Перелік графічного матеріалу:

6. Консультанти розділів проекту:

з технічної частини _____к.т.н. доцент Дорогий Я.Ю._____

7. Дата видачі завдання “___” _____2019_р.

Керівник дипломного проекту _____Я.Ю.Дорогий_____
(підпис)

Завдання прийняв до виконання _____Л.С.Субботіна_____
(підпис)

ЗАТВЕРДЖУЮ
Керівник дипломного проекту
_____ Я.Ю.Дорогий
(підпис) (ініціали, прізвище)
“_ __” _____2019_р.

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК

виконання дипломного проекту

студентом Субботіною Лідією Сергіївною
(прізвище, ім'я, по батькові)

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання	Примітка
1			
2			
3			
4			
4			
5	Оформлення текстової та графічної документації	20.05.2019	
6	Подання проекту до попереднього захисту	04.06.2019	
7	Представлення до захисту		

Студент _____ Л.С.Субботіна
(підпис) (ініціали, прізвище)
Керівник проекту _____ Я.Ю.Дорогий
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Субботіна Л.С. Методи розподілу ресурсів критичної ІТ-інфраструктури. НТУУ «КПІ ім. Ігоря Сікорського», Київ, 2019.

У даному дипломному проекту наведено огляд підходів до побудови корпоративної мережі. Проведено аналіз існуючого обладнання та обґрунтований вибір для реалізації проекту. У якості демонстраційної частини розроблено модель корпоративної мережі на базі протокола MPLS, виконані потрібні налаштування та перевірена її працездатність.

Проект містить 62 с. тексту, 18 рисунків, 1 таблицю, 16 літературних джерел.

Ключові слова: корпоративна мережа, BGP, MPLS, Dynamips, GNS3.

Annotation

Rusilo Yu.M. Secured corporate network based on MPLS technology. NTUU "Igor Sikorsky Kyiv Politechnic Institute", Kyiv, 2019.

This graduation project gives an overview of approaches to building a corporate network. An analysis of the existing equipment and a reasonable choice for the project implementation have been carried out. As a demonstration part, a corporate network model based on the MPLS protocol was developed, the necessary settings were made and its performance tested.

The work contains 62 p. of text, 18 pictures, 1 table, 16 references.

Keywords: corporate network, BGP, MPLS, Dynamips, GNS3.

ЗМІСТ

ВСТУП

Початок XXI століття ознаменувався стрімким розвитком інформаційних технологій. Це зумовило виникнення потреб в розробці обчислювальних центрів для впровадження необхідних бізнес-процесів і при цьому забезпечення мережі для об'єднання інформаційних центрів в одну систему. Так, в подальшому набуває стрімкого розвитку IT – інфраструктура.

Велика кількість економічних, соціальних та стратегічних об'єктів як державного так і приватного сектору мають потребу в обчислювальних ресурсах. Особливо варто приділити велику увагу критичній IT – інфраструктурі, що дає змогу забезпечити високий рівень надійнісості та безпеки для критичних інформаційних процесів. Вимоги до створення та управління критичною IT – інфраструктурою більш жорсткі, оскільки вища вартість ресурсів та важливість інформаційних процесів. Сам процес управління ресурсами вимагає прийняття оптимальних рішень для складних масштабованих систем, що вже є нетривіальною задачею. Проте розробка програмного комплексу для забезпечення оптимального розподілу ресурсів між критичними інформаційними процесами дала б змогу значно зменшити витрати на обслуговування даної IT – інфраструктури та забезпечити максимальну надійність та забезпеченість при проведенні інформаційних операцій.

В даний час широкого поширення набули хмарні технології (хмарні обчислення). Дані технології дають змогу зберігати інформацію децентралізовано, причому виграш є одразу в декількох напрямках, наприклад: надійність, живучість, економічність зберігання. Тому також варто їх розглянути при забезпеченні функціонування критичної IT – інфраструктури.

Метою даного проекту є розробка моделей та методів розподілу ресурсів критичної IT – інфраструктури з використанням хмарних обчислень IaaS, та реалізація даних методів і моделей у вигляді програмної бібліотеки. Частина методів реалізована на мові Python в середовищі PyCharm, а частина на мові C++ в середовищі Visual Studio 2015 з метою збільшення швидкості виконання «важких»

ділянок програмного коду. Програмний пакет призначений для моделювання критичної IT-інфраструктури, а також забезпечення оптимального розподілу ресурсів при обслуговуванні критичних бізнес-процесів та універсальних сервісів.

Об’єктом дослідження є системи управлінням розподілом ресурсів в критичній IT – інфраструктурі. Предметом дослідження є багаторівнева система управління розподілом ресурсів в критичній IT – інфраструктурі за декількома критеріями оптимальності з використанням хмарних обчислень.

Отже, в даній роботі буде розглянуто предметну область використання програмного пакету, проведено аналіз існуючих моделей для розподілу ресурсів критичної IT-інфраструктури, розроблено математичну модель для багаторівневої задачі розподілу ресурсів критичної IT – інфраструктури за декількома критеріями оптимальності. Також описано процес розробки програмного пакету та його тестування.

1 ПОСТАНОВКА ЗАДАЧІ

Розглянемо приклад, коли при наданні сервісу IaaS ресурси віртуального серверу розподіляються нодами, а облік їх використання здійснюється в нодо-годинах. Критичний сервіс вимагає фіксовану мінімальну кількість нод, яка гарантовано буде знаходитись у його розпорядженні в будь який момент часу, і яка буде зарезервована за ним навіть під час простою. Крім того, критичний процес/сервіс має можливість зарезервувати кількість нод, що будуть надані йому додатково, у випадку потреби з його сторони та наявності відповідних ресурсів у хмарі.

Необхідно розробити моделі і методи розподілу ресурсів і навантаження хмарних ЦОД, що відповідають наведеним вище особливостям хмарних IT-інфраструктур, базуються на прийнятних для провайдерів критеріях і враховують ресурсні, технологічні та інші обмеження.

2 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Інформаційна інфраструктура (IT – інфраструктура) – організаційно технічне об’єднання програмних, обчислювальних та комунікаційних засобів, зв’язків між ними, а також обслуговуючого персоналу з метою забезпечення надання інформаційних, обчислювальних та телекомунікаційних ресурсів, сервісів підрозділам організації, що необхідні для впровадження певних бізнес-процесів.

Критична IT-інфраструктура – це сукупність IT-інфраструктур державного та приватного сектору, які забезпечують функціонування та безпеку стратегічних інститутів, систем і об’єктів держави (органів центрального та місцевого управління, систем управління енергетикою, транспортом, зв’язком, банківським сектором, підприємств, під час діяльності яких використовуються та/або виробляються небезпечні речовини тощо) і безпеку громадян (системи управління правоохоронних структур і оборонного сектору тощо), несанкціоноване втручання в роботу яких може загрожувати економічній, екологічній, соціальній та іншим видам безпеки або завдати шкоди міжнародному іміджу держави [1].

Критична IT-інфраструктура також повинна відповідати наведеним вище критеріям, а також має деякі інші критерії та обмеження, що є специфічними для галузі її використання та цілей, які перед нею ставляться, і які будуть розглянуті в цій статті.

При розгляді питань віртуалізації ресурсів критичної IT-інфраструктури виникає ряд додаткових обмежень, які і будуть розглянуті в даній роботі.

Швидкий розвиток інформаційних технологій, їх активне впровадження в процеси управління створили ситуацію, коли сам процес надання інформаційних послуг став об’єктом управління.

Основні постачальники комп’ютерного та телекомунікаційного обладнання й інформаційних технологій представили для широкого загалу велику кількість рішень, призначених для розв’язання проблеми створення IT-інфраструктур. Методологічні засади її розв’язання викладені в рішеннях ITIL, на основі якої отримала розвиток ITSM. Остання розробка, яка досить детально описує

проблематику та майже повністю перекриває попередні розробки – COBIT [2]. Всі вказані розробки є методологічним апаратом, який потрібно використовувати для створення конкретної критичної IT-інфраструктури. Специфіка життя в окремо взятій країні також накладає свої обмеження на створення тієї чи іншої критичної IT-інфраструктури.

На даний момент в Україні вже почалася робота в напрямку розвитку проблематики критичних IT-інфраструктур, хоча ракурс розгляду в основному стосується безпеки функціонування. Теж саме можна сказати і про міжнародних дослідників, хоча деякі роботи [3-5] і розглядають частково інші питання, крім безпеки. Остання розробка [6] дещо покращує розуміння поставленої проблематики, але не надає конкретних алгоритмів щодо планування, створення, управління та інших задач, що виникають при розробці IT-інфраструктур, і тим більше, не відповідають на ці ж самі питання стосовно критичних IT-інфраструктур. Питання віртуалізації підняті в деяких роботах [7-9], але розглянуті з точки зору звичайних IT-інфраструктур, де немає жорстких обмежень та вимог стосовно виділення ресурсів для деяких процесів та сервісів. Нехай маємо деяку критичну IT – інфраструктуру. Для критичної IT - інфраструктури характерна наявність потужних і досить дорогих ресурсів, тому виникає проблема їх ефективного використання. Для цього виконується розподіл, управління та диспетчерування ресурсів і навантаження з використанням певних математичних моделей та методів. Зведення проблеми ефективного використання ресурсів до формального вигляду показало, що необхідно використовувати нечітке програмування з широким вибором критеріїв оптимізації і врахуванням критичності часових, ресурсних, технологічних та інших обмежень.

При формуванні критерію оптимальності для створення і подальшого функціонування критичної IT – інфраструктури оберемо наступну множину параметрів:

- надійність – показник надійності критичної ІТ-інфраструктури в період експлуатації;
- живучість – можливість виконувати свої функції при втраті ресурсів, підсистем і т. ін.;
- забезпеченість – показник максимальної кількості процесів та сервісів, що обслуговуються;
- відновлюваність – тривалість відновлення готовності до експлуатації;
- економічність – витрати різноманітних ресурсів на забезпечення функціонування критичної ІТ-інфраструктури;
- безпечність – показник неможливості виконання несанкціонованих дій, спрямованих на порушення роботи критичної ІТ-інфраструктури чи її частин;
- строк життя;
- ефективність – поєднання вище згаданих параметрів в кожному окремому випадку під визначену задачу.

При формуванні вимог до критерію управління критичною ІТ-інфраструктурою необхідно також враховувати особливості розв’язання задачі. Слід зазначити, що визначення всієї множини параметрів не можна повністю звести до системи формалізованих процедур, бо деякі з них вимагають якісного аналізу. Для такого аналізу слід використати метод структуризації, який дозволяє поділити задачу на підзадачі, визначитись за допомогою експертів або без них з методами розв’язання цих підзадач, обмеженнями використання цих розв’язків та методами поєднання розв’язків.

3 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

3.1 Аналіз існуючих хмарних технологій

3.1.1 Основні напрямки хмарних обчислень

Постачальники хмарних сервісів надають широкі можливості користувачам для впровадження хмарних обчислень. Взагалі серед хмарних обчислень можна виділити три основні моделі (рисунок 3.1):

1. IaaS (Infrastructure as a Service);
2. PaaS (Platform as a Service);
3. SaaS (Software as Service).

Рисунок 3.1 – Моделі обслуговування користувачів при наданні послуг хмарних обчислень

Розглянемо коротко кожну з цих моделей.

IaaS зазвичай надає уніфіковані апаратні і програмні ресурси, але в деяких випадках і на інфраструктурному рівні для установки ПО з оплатою «pay as you go» (по мірі використання). Замовлена інфраструктура може динамічно масштабуватися. На базі такого підходу побудовані Amazon EC2 (Elastic Cloud Computing) Service і Amazon S3 (Simple Storage Service).

PaaS надає більш високий рівень сервісу, що дозволяє розробляти, тестувати і впроваджувати власні програми. Вбудована масштабованість накладає обмеження на тип розроблюваного програмного забезпечення. Яскравим прикладом реалізації такого підходу є сервіс Google App Engine, що дозволяє впроваджувати Web-додатки на тій же системі, на якій працюють власні додатки Google.

SaaS пропонує готове спеціалізоване ПО, що веде до спрощення використання додатків і до зменшення витрат на розробку. Одним з чудових прикладів реалізації за таким підходом є Salesforce та її онлайнова система управління відносинами з клієнтами. Дослідження компанії Forrester показують, що серед різних варіантів хмарних середовищ переважає SaaS [10].

Виконаємо порівняння моделей даних хмарних технологій. Дані для порівняння наведені в таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика моделей хмарних обчислень з різними правами доступа клієнта до обчислювальних ресурсів.

	Власний сервер	Інфраструктура (як сервіс)	Платформа (як сервіс)	Програмне забезпечення (як сервіс)
Застосування	+	+	+	-
Дані	+	+	+	-
Час виконання	+	+	-	-
Проміжний шар	+	+	-	-
О/С	+	+	-	-
Віртуалізація	+	-	-	-
Сервера	+	-	-	-
Сховище	+	-	-	-
Нетворкінг	+	-	-	-

В даній роботі буде розглядатися тип хмарних сервісів IaaS (Infrastructure as a Service), оскільки необхідно мати прямий доступ до апаратних та програмних ресурсів з метою подальшого оптимального їх розподілу.

3.1.2 Хмарні обчислення IaaS

Хмарний сервіс IaaS (інфраструктура як сервіс) являє собою найбільший сегмент ринку хмарних обчислень. В подальшому будуть розглянуті публічні хмарні сервіси IaaS. Публічні хмарні обчислення IaaS в контексті даного дослідження визначаються як стандартизовані, в високій мірі автоматизовані застосування множинної аренди (з великою кількістю користуваів), де обчислювальні ресурси, будучи доповнені мережевими можливостями та функціями сховища, належать та організовуються поставщиком сервісів та пропонуються клієнту за вимогою. Клієнт має можливість самостійного керування цією інфраструктурою з використанням графічного користувацького веб-інтерфейсу, що служить в якості консолі керування IT-операціями в конкретному середовищі оточення. Опціонально може пропонуватися інтерфейс програмних застосувань до інфраструктури (Application Programming Interface (API)).

«Інфраструктура як послуга» (IaaS) – це комплексна IT – інфраструктура, що подається у вигляді послуги. Кожен користувач чи клієнт отримує доступ до частини консолідованого пулу об’єднаних ресурсів для створення та використання власної обчислювальної інфраструктури у відповідності з потребами.

Варто більш детально розглянути переваги моделі IaaS перед іншими моделями. Насамперед це:

- Підвищена ефективність – віртуалізовані ресурси об’єднуються в пули, забезпечуючи використання всієї ємності фізичної інфраструктури;
- Підвищення оперативності – IT-ресурси можна виділяти за вимогою та швидко повертати назад в пул;
- Швидке масштабування – миттєве виділення додаткових ресурсів у віповідності з бізнес-вимогами в періоди пікових нагрузок, а також при

збільшенні чи зменшенні розміру організації;

- Зниження затрат – модель «оплата по мірі використання» дає змогу скоротити витрати на інфраструкту, електроенергію та обслуговування;
- Підвищення продуктивності роботи IT – служби – автоматизоване виділення ресурсів через портал самообслуговування;
- Скорочення невикористовуваних ресурсів – прозорі методи ціноутворення, вимірювання та розподілу витрат між підрозділами дають змогу IT-адміністраторам виявити потенційні області скорочення затрат;
- Підвищення ефективності інвестицій в IT-інфраструктуру;
- Підвищення рівня безпеки та захисту інформаційних ресурсів.

3.1.3 Порівняльна характеристика основних провайдерів IaaS

На ринку послуг IaaS найбільної популярності здобули такі провайдери, як: Amazon Web Service EC2, Google Compute Engine та Microsoft Azure. Для них ми і виконаємо порівняльну характеристику [11].

3.1.3.1 Провайдер Amazon Web Service

AWS включає в себе обслуговування ядра обчислень Amazon, що надає змогу користувачам налаштовувати віртуальні машини з використанням попередньо сконфігурованих (за замовчуванням) чи користувацьких MAC (шаблонів віртуальних машин). В даному випадку користувач має можливість обрати розмір, потужність, обсяг пам’яті і число віртуальних машин з різних регіонів та призначити для них зони доступності, в межах яких можна запускати дані машини. AWS надає також можливості балансування навантаження (ELB) та автоматичного масштабування. ELB розподіляє навантаження між екземплярами машин для підвищення продуктивності.

APM забезпечує ефемерне (тимчасове) зберігання, при якому створюється лише один екземпляр сеансу і який руйнується при завершенні роботи. AWS також

пропонує зберігання об'єктів за допомогою S3 служби та послуги архівування. AWS повністю підтримує реляційні і NoSQL бази даних, великі обсяги даних.

Віртуальні приватні хмари Amazon (VPCs) дозволяють користувачам створювати групи віртуальних машин в ізольованих мережах в хмарі. За допомогою VPCs користувачі можуть визначати топологію мереж, створювати підмережі, таблиці маршрутизацій, приватні діапазони адрес і мережеві шлюзи.

Amazon Web Service має досить зручний інтерфейс для контролю за станом хмари, кількістю розгорнутих віртуальних машин, логічних дисків і т.п. Приклад інтерфейсу Amazon Web Service наведено на рисунку 3.2.

AWS приваблює клієнтів методом оплати за використання ресурсів хмари. Користувач платить тільки за «округлену» кількість годин, впродовж яких він використовував дану систему. Мінімальна одиниця часу використання – одна година.

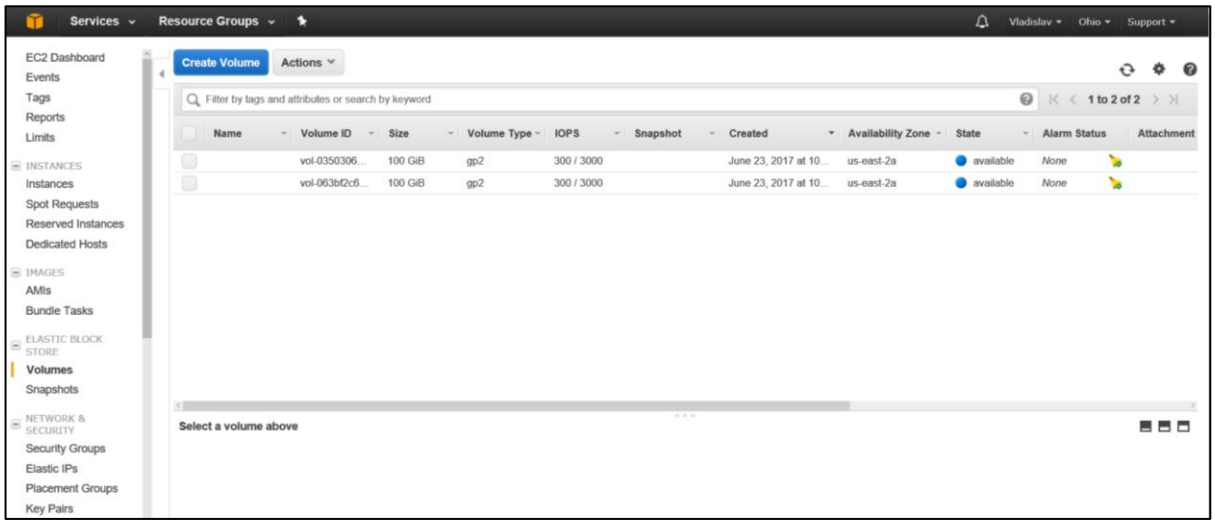

Рисунок 3.2 – Інтерфейс користувача в Amazon Web Service

3.1.3.2 Провайдер Google Compute Engine

У 2012 році компанія Google представила свій хмарний сервіс обчислень – Google Compute Engine (GCE). Google Compute Engine дозволяє користувачам запускати віртуальні машини так само, як AWS, в різних регіонах та зонах доступності. Однак, GCE не була широко доступна для користувачів до 2013 року.

Лише з 2013 року Google почала проводити свої власні удосконалення, такі як балансування навантаження, розширена підтримка операційних систем, жива міграція віртуальних машин. Інтерфейс користувача для Google Compute Engine наведено на рисунку 3.3.

Хмарна платформа Google, так само забезпечує як тимчасове, так і постійне зберігання даних. Для зберігання об'єктів, GCP має Google Cloud Storage. GCP підтримує реляційну БД через Google Cloud SQL. Nearline Google пропонує архівування практично без затримки на відновлення даних.

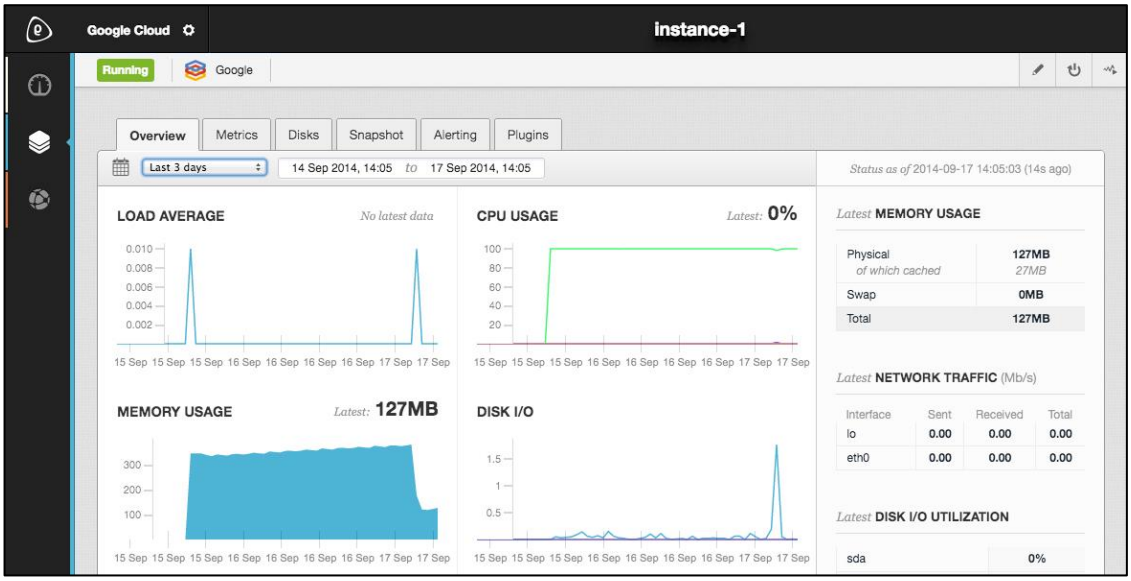

Рисунок 3.3 – Інтерфейс користувача в Google Compute Engine

3.1.3.3 Провайдер Microsoft Azure

У 2012 році компанія Microsoft представила свою службу обчислень, для попереднього тестування, але не зробила її загальнодоступною, доки користувачі Azure в травні 2013 року не обрали VHD (Virtual Hard Disk), що дуже схоже на AMI Amazon, при створенні віртуальних машин. Віртуальний жорсткий диск може бути визначений за замовчуванням, або самим користувачем. Для кожної віртуальної машини необхідно вказати кількість ядер і об’єм виділеної пам’яті. Інтерфейс користувача в Microsoft Asure наведено на рисунку 3.4.

Azure використовує тимчасове сховище (D диск) і Page Blobs (опція Block Storage від Microsoft) для томів віртуальних машин. Блок Blobs і файли служать для зберігання об'єктів. Azure підтримує реляційні, NoSQL бази даних і великі дані, за допомогою таблиць Windows Azure і HDInsight.

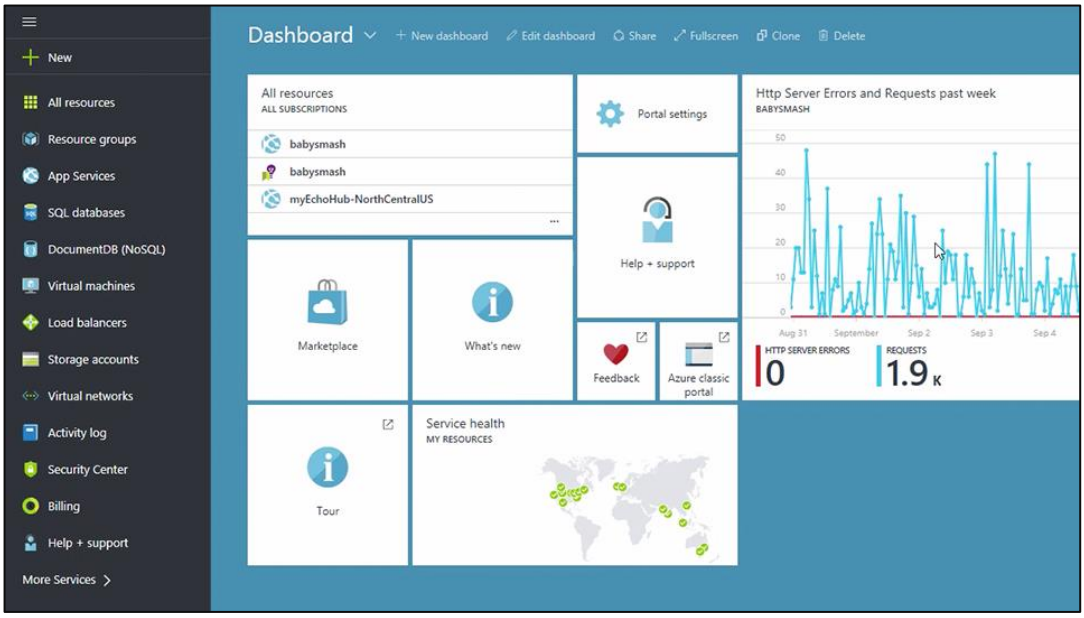

Рисунок 3.4 – Інтерфейс користувача в Microsoft Asure

Результати порівняння провайдерів наведені в таблиці 3.2.

Таблиця 3.2 – Порівняльна характеристика найпопулярніших провайдерів IaaS

Продовження таблиці 3.2

	Amazon Web Services (AWS)	Microsoft Azure	Google Compute Engine
Кількість типів віртуальних машин	38	33	18
Кількість сімейств віртуальних машин	7	4	4
Наявність регіонів	+	+	+
Наявність зон	+	-	+
Ціноутворення	За годину, округлюється.	За хвилину, округюється (з передплатою або щомісяця)	За хвилину, округлюється (мінімум 10 хв)

Моделі оплати	На вимогу, зарезервовано за місцем.	За запитом – короткострокові зобов'язання (оплачені або щомісяця)	На вимогу – стале використання
Віртуальна мережа	VPC	VNet	Підмережа
Публічний IP	+	+	+
Гібридні хмари	+	+	-
DNS	Маршру-тизується	-	-
Firewall/ACL	+	+	+
Зберігання об’єктів	S3	Block Blobs and Files	Google Cloud Storage
Блок зберігання	EBS	Page Blobs	Стійкі диски

Нині на ринку хмарних обчислень ці три поставщики послуг є найбільшими конкурентами, хоча Amazon Web Services має значну фору. Тому оберемо цього провайдера хмарних обчислень для подальшо використання під даний проект

3.1.4 Веб-сервіс Amazon Web Service EC2

Amazon Elastic Compute Cloud (Amazon EC2) – веб-сервіс, котрий надає користувачу обчислювальні потужності в хмарі. Даний сервіс входить в інфраструктуру Amazon Web Services.

Даний сервіс має простий веб-інтерфейс, що дозволяє отримати доступ до обчислювальних потужностей та налаштувати ресурси з мінімальними затратами за короткий час. Користувач має повний контроль за обчислювальними ресурсами, а також доступне середовище для роботи.

За допомогою EC2 користувач може:

- створювати Amazon Machine Image (AMI), котрий буде утримувати програми, бібліотеки, дані та та пов'язані з ними функціональні параметри або використовувати раніше налаштовані шаблони образів та робіт;
- завантажити AMI в Amazon S3. Amazon EC2 пропонує інструменти, для зберігання AMI. Amazon S3 забезпечує захищене, надійне та швидке сховище для зберігання образів;
- використовувати Amazon EC2 веб-сервіс для налаштування безпеки та мережевого доступу;
- обирати тип операційної системи, за власними потребами, запускати, вимикати або контролювати декілька AMI по мірі необхідності, використовувати API веб-сервісу, або різних інструментів управління, які раніше були передбачені;
- визначити необхідність локалізації в декількох місцях, використовуючи статичні IP-адреси або інші варіанти;
- оплачувати тільки за ресурси, котрі будуть використовуватись, такі як час або кількість переданої інформації (кількість трафіку).

3.2 Аналіз існуючих моделей розподілу ресурсів критичної IT-інфраструктури

3.2.1 Модель розподілу ресурсів IT-інфраструктури з чіткими параметрами

Для впровадження моделі розподілу ресурсів необхідно розглянути декілька ознак, що можуть вплинути на тип самої моделі [12]. Моделі можуть відрізнятися в залежності від методів ведення бізнесу (для власних бізнес-процесів, чи для зовнішніх клієнтів). Також на вибір моделей впливає тип архітектури IT-інфраструктури. Залежно від рівня абстракції ресурсів варіюється складність математичної моделі задачі, що розв’язується. Задачі великої розмірності

розв’язуються у два етапи: на першому етапі здійснюється розподіл абстрактних ресурсів кожного типу без прив’язки до їх конкретного місцезнаходження з уточненням отриманих результатів на другому етапі.

Дуже істотно впливає на вибір моделей ознака забезпеченості ресурсами, тобто, чи дозволяється часткова підтримка сервісів, чи вони мають підтримуватися у повному обсязі, або не підтримуватися взагалі. У найпростішому випадку дворівневої клієнт-серверної архітектури модель розподілу обмежених ресурсів може подаватись у вигляді декількох бізнес-процесів, що безпосередньо використовують частину одного чи декількох незалежних один від одного ресурсів (рисунок 3.5).

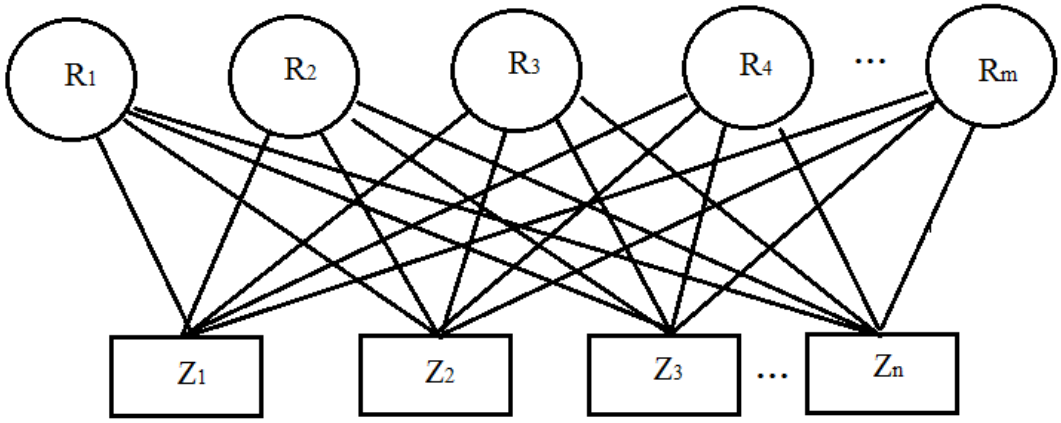

Рисунок 3.5 – Модель розподілу обмежених ресурсів IT-інфраструктури

Введемо необхідні для побудови моделей наступні позначення:

- $Z_{1,\dots,}Z_n$ – бізнес-процеси, підтримку яких забезпечує функціонування ІТС;
- $W = (w_1, \dots, w_n)$ – коефіцієнти важливості бізнес процесів $Z_{1,\dots,}Z_n$ відповідно;
- $R_{1,\dots,}R_m$ – інтегровані ресурси ІТС, необхідні для підтримки функціонування бізнес-процесів;
- P = $\|p_{ij}\|$ – матриця потреб бізнес-процесів у ресурсах ІТС, де p_{ij} відповідає наведеній кількості потрібного для процесу Z_i ресурсу R_j чи 0, якщо ресурс не потрібен;
- D = $\|d_{ij}\|$ – матриця наявності потреб бізнес-процесів у ресурсха ІТС, де:

$$\begin{cases} d_{ij} = 1, якщо\ p_{ij} > 0 \\ d_{ij} = 1, якщо\ p_{ij} = 0 \end{cases} \quad (3.1)$$

- R = $(r_1, \dots, r_m)$ – вектор встановлених обмежень на ресурси.

Розглянемо дискретний випадок, коли бізнес-процес або підтримується в повному обсязі, або повністю блокується. Введемо вектор X = $(x_1, \dots, x_n)$, де:

$$x_i = \begin{cases} 1, якщо\ процес\ Z_i\ обслуговується \\ 0, в\ протилежному\ випадку \end{cases} . \quad (3.2)$$

Тоді критерій оптимального розподілу ресурсів ІТС можна подати у вигляді:

$$\max \sum_{i=1}^{n} x_i w_i. \quad (3.3)$$

Мають також бути використані обмеження по ресурсам (3.4)

$$\sum_{i=1}^{n} x_i p_{ij} \leq r_j, j = 1, \dots, m. \quad (3.4)$$

Варто зазначити, що наведені вище моделі, а також інші моделі, які розглядаються, належать до лінійних або нелінійних, неперервних, булевих та змішаних задач математичного програмування, які до того ж мають стохастичні аналоги. Для детермінованих задач можна використовувати евристичні алгоритми, методи м'яких обчислень та точні методи, використання яких обмежено з огляду на розміри задач. Так, для задач лінійного програмування можна використовувати відомі методи. Задачі булевого програмування можна розв'язувати методами часткового перебору. Точні методи дають змогу знайти найкращий розв'язок, але їх використання можливе лише для задач обмеженої розмірності, оскільки час пошуку рішень суттєво зростає зі збільшенням складності задачі. На відміну від

них, евристичні методи та методи штучного інтелекту, насамперед генетичні алгоритми (ГА), дають змогу знайти задовільний розв'язок за доволі короткий час, навіть для задач великої розмірності, до того ж їхню ефективність можна поліпшити за рахунок врахування особливості задач. Загальна схема алгоритму подана на рисунку 3.6.

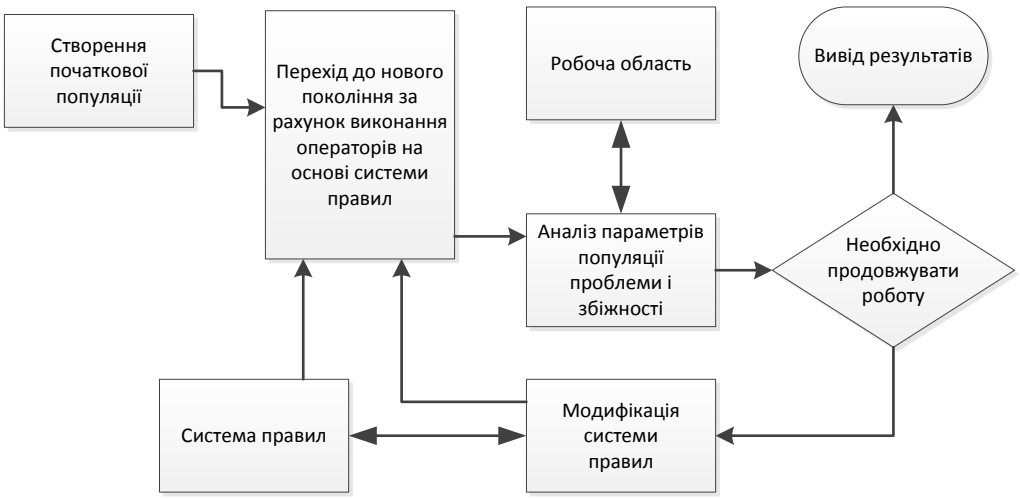

Рисунок 3.6 – Схема роботи генетичного алгоритму

Використання агентів дає змогу здійснювати управління ресурсами і навантаженням на двох рівнях. Перший рівень – управління окремими віртуальними серверами. Другий рівень – управління окремими застосуваннями, що встановлені на віртуальних серверах. Загальну схему алгоритму наведено на рисунку 4. Модуль прогнозування призначений для аналізу трендів з метою подальшої реалізації проактивного управління, у цьому випадку – перерозподілу ресурсів між застосуваннями до того, як виникла їх нестача для застосувань, задіяних у підтриманні бізнес-процесів високої важливості. Модуль планування призначений для довгострокового управління ресурсами IT-інфраструктури у разі її розвитку чи істотних змін у її функціонуванні. Диспетчер запитів слугує для обмеження клієнтського трафіку заданого типу з метою вивільнення ресурсів, споживання яких залежить від навантаження.

3.2.2 Модель управління віртуальними машинами при серверній віртуалізації

Нехай є декілька фізичних серверів $S_i, i = 1, \dots, n$, на яких під управлінням гіпервізорів функціонують віртуальні машини (ВМ) $V_j, j = 1, \dots, m$ (рисунок 3.7).

Кожна з ВМ в залежності від потоку клієнтських запитів використовує певну кількість ресурсів типу $R_k, k = 1, \dots, l$ (пам'ять, процесорний час, дисковий простір, пропускна спроможність підсистеми вводу-виводу, зовнішніх каналів зв'язку тощо) .

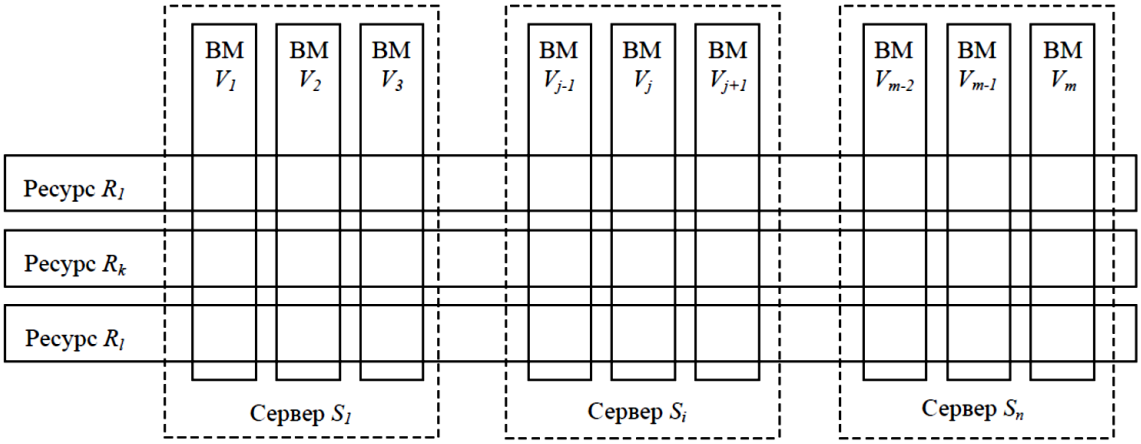

Рисунок 3.7 – Модель розміщення віртуальних машин на фізичних серверах

Розглянемо процес розподілу ресурсів серверів між віртуальними машинами [13]. Для цього введемо необхідні для побудови моделей позначення:

- r_{kj}– кількість ресурсу типу R_k, що встановлено на сервері S_i;
- p_{kj} – потреби ВМ V_j у ресурсах типу R_k для забезпечення заданого рівня SLA;
- x_{ij} – булева змінна, яка визначає, чи встановлена ВМ V_j на сервері S_i.

Оскільки кожна ВМ одночасно розташована не більше ніж на одному сервері, має виконуватись умова описана у формулі 3.5:

$$\sum_{i=1}^{n} x_{ij} \leq 1, j = 1, \dots, m \qquad (3.5)$$

До того ж, для нормального функціонування ВМ повинні бути забезпечені достатнім обсягом ресурсів серверів, на яких вони розташовані (формула 3.6).

$$\sum_{i=1}^{m} x_{ij}\, p_{kj} \le r_{ki}, k = 1, \dots, l; i = 1, \dots, n. \qquad (3.6)$$

Розглянемо задачу розташування ВМ для випадків нестачі та надлишку ресурсів.

У разі, якщо внаслідок збільшення клієнтських запитів потреба окремих ВМ у ресурсах певного типу збільшилась настільки, що стає неможливим забезпечити усі ВМ необхідною кількістю ресурсів, природним виходом стає задача підтримки найбільш важливих бізнес-процесів шляхом забезпечення ресурсами тих ВМ, на яких працюють пов'язані з ними сервіси, за рахунок менш важливих.

Позначимо через $w_j, j = 1, \dots, m,$ важливість застосувань, встановлених на ВМ V_j. Тоді задачу можна сформулювати наступним чином (3.7).

$$\sum_{j=1}^{m}\sum_{i=1}^{n} x_{ij} w_j \rightarrow max, \qquad (3.7)$$

при обмеженнях (3.5), (3.6).

Ця задача становить собою лінійну задачу булевого програмування. Для її вирішення скористаємося методом гілок та меж. Але спочатку покажемо, як можна, враховуючи специфіку задачі, зменшити початкову кількість можливих комбінацій, яка складає 2^{mn}.

По-перше, згідно з обмеженням (3.5), кожна ВМ може бути розташована не більше, ніж на одному сервері, тобто замість перебору 2^n комбінацій для кожної ВМ Vj, можна обмежитись розглядом лише n + 1 варіантів: y_j = 0,..., n, де y_j – номер сервера, на якому встановлена ВМ V_j (y_j = 0 у випадку, якщо внаслідок

нестачі ресурсів ВМ V_j не встановлена на жодному сервері або ресурси для неї виділяються по залишковому принципу).

По-друге, враховуючи те, що вимоги ВМ у ресурсах p_{ij} є невід'ємними, якщо на якомусь з етапів побудови дерева варіантів розміщення ВМ по серверам одне з обмежень (3.6) перестає виконуватись, продовжувати побудову цієї гілки немає сенсу, оскільки для усіх її вузлів це обмеження також не буде виконуватись.

По-третє, слід врахувати те, що дуже часто при побудові серверних ферм використовують сервери з однаковою конфігурацією. Якщо, наприклад, є два ідентичних сервера, кількість можливих варіантів розміщення ВМ можна скоротити в 2 рази, оскільки перенесення усіх ВМ з першого сервера на другий, а з другого на перший ніяк не вплине на об'єм ресурсів, які використовуються. Тобто перестановка рядків у матриці $X = \|x_{ij}\|$,які відповідають однаковим стовпцям у матриці $X = \|x_{ij}\|$, ніяк не впливає ані на виконання обмежень (3.5), (3.6), ані на значення критерію (3.7). Іншими словами, для серверів з однаковою конфігурацією важливе не абсолютне розміщення ВМ на цих серверах, а їх розташування одна відносно іншої. При збільшенні кількості серверів з однаковою конфігурацією кількість «зайвих» комбінацій значно зростає від (n-1) для випадку розміщення усіх ВМ на одному сервері до (n!-1) при розміщенні ВМ на різних серверах.

Так, наприклад, для випадку трьох ВМ і трьох ідентичних серверів з можливих 27 комбінацій розташування ВМ лише 5 є унікальними, а решта – 22 комбінації – «дзеркальними» до них. Тобто, навіть для таких незначних значень m і n початкову множину варіантів можна скоротити більше, ніж у 5 разів.

Для наочності буквами («A», «B», «C») позначені ВМ, цифрами (1, 2, 3) – номера серверів, комбінацією букви з цифрою – розміщення ВМ на сервері. Алгоритм перебору при побудові дерева розташування ВМ полягає у наступному: на черговому рівні чергової гілки кількість серверів, що розглядаються, для перебору приймається на 1 більше, ніж максимальна для попередніх рівнів цієї ж гілки, але не більше ніж n.

Так, на рівні «A» розглядається лише випадок розташування ВМ «A» на сервері №1, а на серверах №2 та №3 не розглядається, оскільки сервери №1-3 є

повністю ідентичними і розташування ВМ «А» на них призведе до тих самих результатів. При розв'язанні задачі в реальних умовах, з метою зменшення кількості міграцій ВМ, при виборі довільного сервера обирається той, на якому ця ВМ вже встановлена. На рівні «В» розглядаються два варіанти «В1» – розміщення ВМ «В» на тому ж сервері, що і ВМ «А», та «В2» – розміщення ВМ «В» на іншому сервері, у цьому випадку №2. Варіант «В3» не розглядається, оскільки він нічим принципово не відрізняється від варіанту «В2» і призведе до тих самих результатів (аналогічно, якщо ВМ «В» уже встановлена на сервері №3, слід ви- користовувати саме його, а не сервер №2).

Якщо у наявності є сервери з іншою конфігурацією, до схеми перебору додаються відповідні вузли на кожному рівні. Сюди також відносяться випадки, коли внаслідок нестачі ресурсів дозволяється низькопріоритетні ВМ не розміщувати на жодному сервері, чи виділяти ресурси для них за залишковим принципом. Формально це описується шляхом розміщення таких ВМ на неіснуючому сервері № 0.

Крім визначення процедури розгалуження метод гілок і меж передбачає визначення процедур оцінки верхньої та нижньої меж за допомогою наближених, але швидких алгоритмів.

При вирішенні задача максимізації для визначення оцінки зверху можна використати оптимістичний прогноз, який полягає в припущенні того, що на черговому кроці алгоритму усі ВМ, не розподілені між серверами, можуть бути розподілені таким чином, що жодна з них не отримує відмови в обслуговуванні (не буде розміщена на неіснуючому сервері № 0) за умови виконання ресурсних обмежень (формула 3.6).

Щоб одержати оцінку знизу, пропонується використати жадібний алгоритм, упорядкувавши ВМ за зменшенням вимог до того ресурсу, нестача якого стала причиною переходу від старого до нового плану розміщення ВМ.

Процедура відсікання гілок полягає у тому, що гілка, для якої оцінка верхньої межі менша, ніж оцінка нижньої межі хоча б однієї з інших гілок, розглядається як безперспективна та виключається з дерева рішень. Враховуючи те, що вимоги ВМ

до ресурсів є невід’ємними, ця процедура може бути до- повнена відсіканням тих гілок, для яких припиняють виконуватись обмеження (3.6).

У разі, якщо на черговому кроці буде знайдене рішення, при якому усі ВМ успішно розміщені, тобто жодна ВМ не розмішена на сервері № 0, пошук можна припинити, оскільки глобальний оптимум знайдений і критерій (3.7) набуває максимального значення. Але, за умови наявності часових можливостей, продовження пошуку може допомогти знайти рішення, що забезпечить меншу, кількість міграцій ВМ при реалізації нового плану розташування, ніж уже знайдене рішення та його «дзеркальні» рішення.

4 ОПИС АРХІТЕКТУРИ ХМАРИ ТА ПРОЦЕСУ ОБСЛУГОВУВАННЯ БІЗНЕС-ПРОЦЕСІВ І КРИТИЧНИХ СЕРВІСІВ

4.1 Опис архітектури хмари

Розглянемо розподілену та багаторівневу хмару. Зазвичай, така хмара має сотні серверів, розташованих на різних стійках, в різних географічних точках. З'єднання між двома машинами з різних стійок може проходити через один або кілька комутаторів. Багаторівневий розподіл представляє дуже складне завдання надійного, масштабованого, доступного поширення даних. Зазвичай центри обробки даних, що представляють собою структуру хмари можуть локалізуватися в декількох географічних точках (рисунок 4.1). Фізичні сервери в межах датацентру розподілені між стійками (рисунок 4.2). Детальніше архітектура хмари наведена в діаграмі кресленика Д1.

Рисунок 4.1 – Карта локалізації датацентрів в різних географічних точках світу

Рисунок 4.2 – Розташування стійок з серверами в межах датацентру

4.2 Сервісно-орієнтована архітектура

В даному проекті розглядається проблема обслуговування універсальних сервісів та бізнес-процесів.

Як правило, існуючі корпоративні застосування складаються з певного числа монолітних модулів, в кожен з яких часто включають реалізацію однакових фрагментів бізнес-логіки. При зміні певного фрагменту бізнес логіки, необхідно вносити зміни до всіх інших фрагментів, що є дуже затратною та ненадійною операцією. При розміщенні даних застосувань в хмарі виникає проблема, пов'язана з неефективним використанням ресурсів, тому все більше популярності набирає сервісно-орієнтована архітектура (СОА) з використанням відкритих стандартів.

Сервіс – незалежний програмний компонент, що здатний виконувати певну задачу, і не потребує для використання клієнтами певної програмної технології.

Використання сервісів значно зменшує час підключення новго бізнес-сервісу до існуючої системи. Сервіси дають змогу реалізувати повторювані бізнес-функції для багаторазоваого використання, що необхідні для організації злагодженої роботи складних застосувань, і які складаються з великого числа різних компонентів.

Внісши зміни лише до одного сервісу ми тим самим змінюємо поведінку декількох бізнес-процесів, що залежить від даного сервісу. Це і являє суттєву перевагу СОА.

Низька зв'язаність – важливий архітектурний принцип при розробці СОА – систем. Використання даного принципу дозволяє зв'язувати різні компоненти інформаційної системи під час її функціонуванн за допомогою так названого, пізнього звязування. Завдяки цій особливості також значно полегшується внесення змін функціональність сервісів, оскільки це зовсім не чіпає інші сервіси.

Завдяки низькій зв’язаності значно спрощується покрокове створення корпоративної системи через відсутність бар’єрів реалізації функціональності сервісу за декілька ітерацій.

4.3 Опис роботи системи з управління розподілом ресурсів в хмарі

Для хмари потрібно розробити політику розташування реплік, яка повинна задовольнити наступним властивостям:

- максимізація надійності;
- максимізація забезпеченості.

Репліки повинні бути розташовані не тільки на різних дисках або різних машинах, але і в різних стійках. Це гарантує, що процес або сервіс буде доступним, навіть якщо ціла стійка пошкоджена або відключена від мережі. При такому розташуванні читання займає час, який приблизно дорівнює пропускній здатності мережі, хоча потік даних при записі повинен пройти через різні стійки.

Одночасно, при створенні нового критичного процесу або сервісу, визначається, де розмістити репліку. При цьому потрібно враховувати наступне: Нова репліка критичного процесу або універсального сервісу розміщується на віртуальну машину сервера з найменшою середньою завантаженістю дисків. У такий спосіб вирівнюється завантаженість дисків на різних серверах та досягається найбільша надійність операції.

- Число нових створюваних критичних процесів або універсальних сервісів на кожній віртуальній машині серверів є обмеженим. Незважаючи на те, що створення нового процесу є швидкою операцією, вона передбачає подальший запис даних на цю віртуальну машину, що вже є важкою операцією, і яка може привести до розбалансування обсягу трафіку даних на різні частини системи.
- Як сказано вище, потрібно розподілити віртуальні машини між серверами в різних стійках. Це також дозволяє досягти найбільшої надійності операції.
- Як тільки число реплік падає нижче встановлюваної користувачем величини, потрібно знову виконати репліку критичного процесу або сервісу. Ця ситуація може статися з декількох причин:
    1) віртуальна машина стала недоступною;
    2) сервер став недоступним;
    3) один з дисків вийшов з ладу;
    4) збільшене число реплік.
- Кожному критичному процесу або універсальному сервісу, для якого потрібно зробити репліку, встановлюється відповідний пріоритет, який теж залежить від декількох факторів. По-перше, пріоритет вище у того критичного процесу або універсального сервісу, який має найменше число реплік. По-друге, щоб збільшити надійність виконання застосувань, збільшується пріоритет у процесів або у сервісів, які блокують прогрес у роботі клієнта. В першу чергу, обслуговуються критичні процеси.
- Вибирається процес або сервіс з найбільшим пріоритетом і копіюється з однієї з реплік серверу, який є найбільш забезпеченим. Нова репліка розташовується, виходячи з тих же причин, що і при створенні.
- Створення реплік постійно балансується. Залежно від розподілу реплік в системі, репліки переміщуються для вирівнювання завантаженості дисків і балансування навантаження. Також потрібно постійно вирішувати, яку з реплік має сенс видалити в даний момент. Як правило, видаляється репліка, яка знаходиться на віртуальній машині серверу з найменшим вільним місцем

на жорстких дисках і яка належить найбільш забезпеченому і надійному критичному процесу.

Розглянута модель управління ресурсами (відповідно, ресурсами є кількість необхідних серверів, віртуальних машин та жорстких дисків, кількість вільних ресурсів для створення нових процесів та сервісів, реплік для розміщення кожного з наших критичних процесів та універсальних сервісів з метою їх надійного функціонування) в рамках існуючих обмежень (наявність серверів та місця для розміщення реплік на окремих серверах та віртуальних машинах, кількість наявних ресурсів для створення нових процесів та сервісів) дозволяє розподілити максимальну кількість критичних бізнес-процесів та універсальних сервісів таким чином, щоб не порушувались принципи функціонування розглянутої хмари, гарантувалась максимальна забезпеченість та надійність функціонування такої критичної IT-інфраструктури.

Кількість ресурсів на кожному сервері обмежена, тому їх використання має бути збалансовним з метою збільшення продуктивності. У випадку перезагрузки ресурсу він стає джерелом простоїв в функціонуванні системи. Така ситуація відома як «вузьке місце».

В нашому випадку ми не маємо допустити перевантаження ресурсів при виконанні бізнес процесів та сервісів. Можна тільки обмежети кількість некритичних процесів. Діаграма розподілу ресурсів критичної IT-інфраструктури наведена на кресленику Д2.

4.4 Підсистема видачі реплік на розміщення

Заяви на видачі реплік бізнес-процесів та універсальних сервісів зберігаються в хмарі з певним пріорітетом. Пріорітет видачі заяв на розміщення реплік залежить від двух факторів

- Пріорітет вище у репліки для бізнес-процесу або універсального сервісу з найменшою кількістю реплік у хмарі;
- Пріорітет вище у репліки критичного процесу.

Отже, сумарний пріорітет видачі реплік залежить від двох факторів одночасно. Для організації такої видачі реплік сформуємо чергу з пріорітетом. Місце репліки в черзі визначає число, що формується при перемноженні коефіцієнта критичності на кількість реплік даного бізнес-процесу чи сервісу.

Отже, репліка при попаданні в чергу може одразу «проскочити» декілька позицій в черзі. Даний спосіб видачі пріорітету дійсний як для реплік нових бізнес процесів та сервісів, так і для реплік уже існуючих.

При видачі репліки з вершини черги, та її розміщенні в хмарі, для бізнес процесу або універсального сервісу інкрементується кількість існуючих реплік. Внаслідок цього змінюється пріорітет, і заява на видачу репліки переміщується в сторону хвоста черги. Для кожного бізнес процесу і універсального сервісу є свої коефіцієнти критичності, що встановлюються за нечіткою шкалою (рисунки 4.4, 4.5).

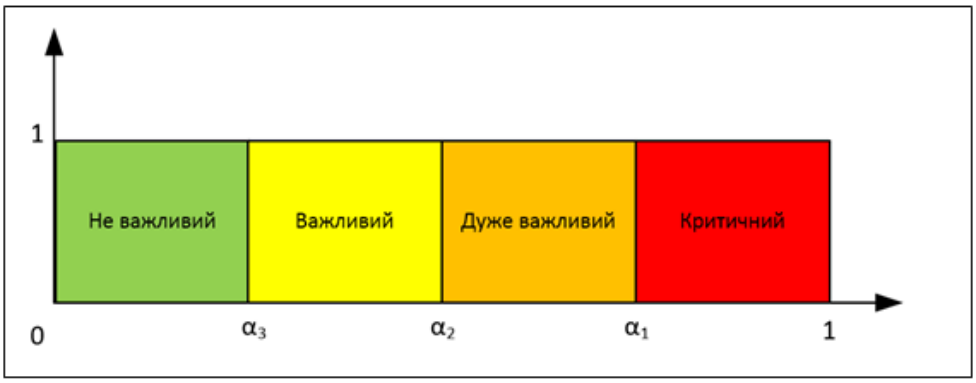

Рисунок 4.3 – Нечітка шкала для визначення рівня критичності бізнес-процесів.

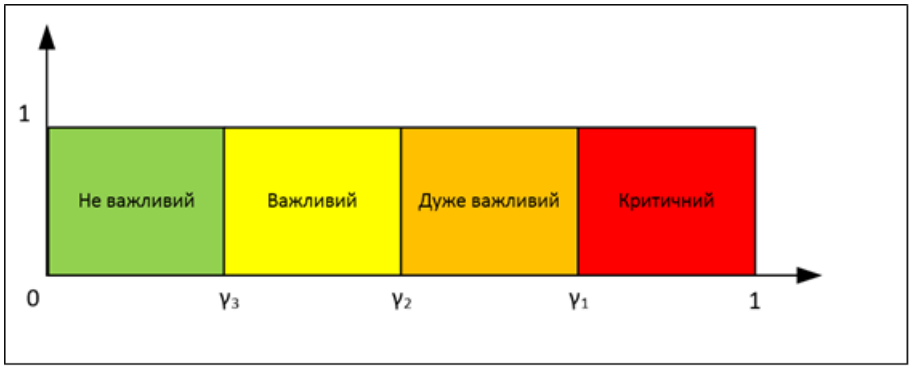

Рисунок 4.4 – Нечітка шкала для визначення рівня критичності універсальних сервісів

Для розміщення репліки необхідно обрати віртуальну машину сервера з найменшою середньою завантаженістю дисків. При цьому необхідно перевірити, чи не перевищений ліміт реплік на даній віртуальній машині. Якщо ліміт перевищений, то необхідно обрати наступний сервер з найменшою середньою завантаженістю дисків, і для нього виконати перевірку на ліміт реплік. При цьому бажано розміщати репліки не тільки на різних дисках або різних серверах, а й на різних серверах. Це підвищить надійність виконання операцій.

4.5 Підсистема балансування реплік

Час від часу на хмарі запускається процес балансування реплік. Залежно від розподілу реплік в системі, репліки переміщуються для вирівнювання завантаженості дисків і балансування навантаження.

Підсистема балансування реплік перерозподіляє репліки між віртуальними машинами, й одночасно віртуальні машини між серверами (жива міграція), бажано на різних стійках. В даному випадку репліки будуть розподілені з максимальною надійністю та забезпеченістю. Процес динамічного перерозподілу віртуальних машин між серверами може підтримуватися не всіма гіпервізорами. Проте більшість нових версій гіпервізорів підтримують дану можливість.

Це дає змогу без втрат продуктивності переміщувати віртуальні машини не тільки в межах стійки чи датацентру, а й декількох регіонів.

5 МАТЕМАТИЧНА МОДЕЛЬ

5.1 Нечіткі числа та операції над ними

Нечітке число - це нечітка підмножина універсальної множини дійсних чисел, що має нормальну і опуклу функцію приналежності.

В даній роботі використовується декілька типів нечітких чисел:

- чіткі числа (як підмножина нечітких);
- інтервальні числа (прямокутні);
- триангулярні числа;
- трапезоїдні числа.

Вигляд функцій приналежності даних чисел наведено на рисунку 5.1.

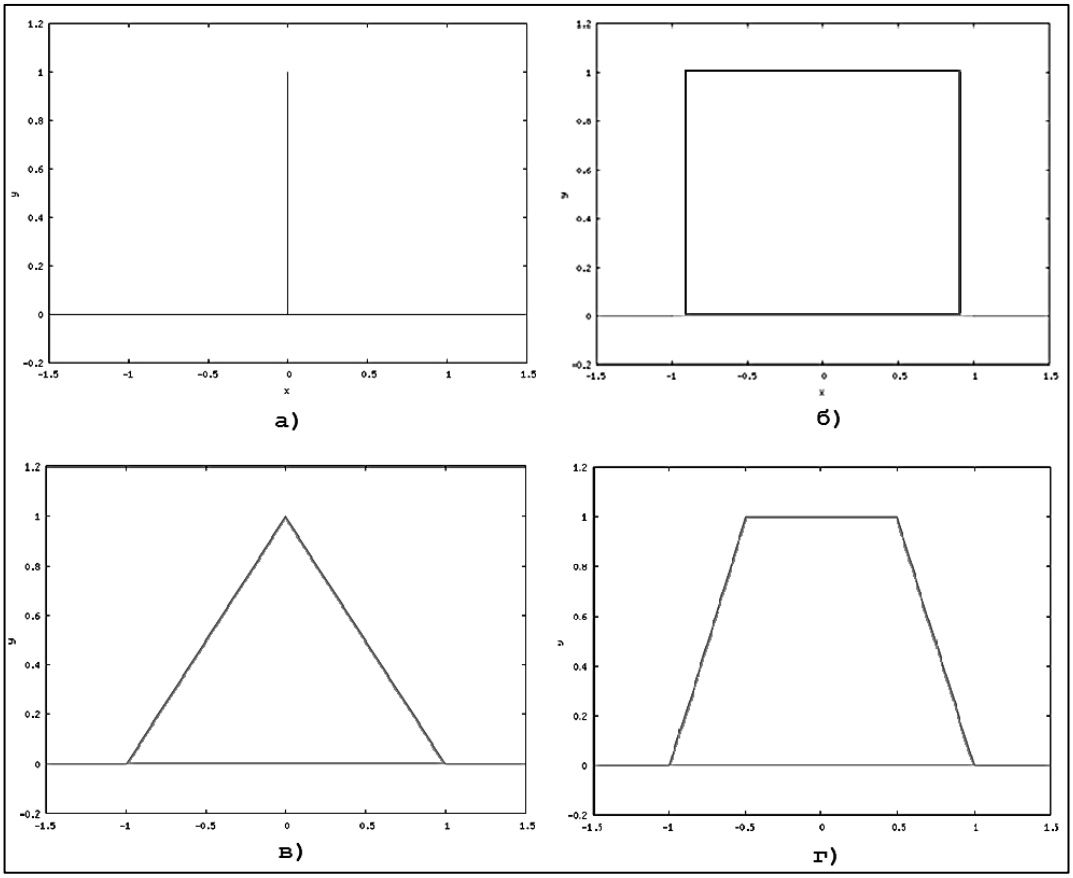

Рисунок 5.1 – Графіки функцій приналежності для нечітких чисел: а) просте число; б) інтервальне число; в) триангулярне число; г)трапезоїдне число

Для зручночті виконання операцій над нечіткими числами триангулярні та трапезоїдні числа представляються в L-R – вигляді.

Дамо визначення для різних типів нечітких чисел, наведемо їх приклади та визначимо базові операції над ними з метою подальшої їхньої обробки.

Визначення 1. Функція, що зазвичай позначається як L (лівостороння) або R (правостороння), є функцією належності нечітких чисел тоді і тільки тоді, коли L(x) = L(-x), L(0) = 1, і L незростає на інтервалі [0, + ∞). Аналогічно, правостороння функція R (·) визначається як лівостороння L (·)[15].

Визначення 2. Триангулярне нечітке число $\widetilde{M}$ називається L-R нечітким числом, якщо існують функції належності L (для лівої частини), R (для правої частини), та числа α > 0, β > 0 такі, що виконується умова (5.1) [15].

$$\mu_{\widetilde{M}}(x) = \begin{cases} L\left(\dfrac{m-x}{\alpha}\right), & x \le m, \\ L\left(\dfrac{x-m}{\beta}\right), & x \le m, \end{cases} \qquad (5.1)$$

де

m – найімовірніше значення числа $\widetilde{M}$,

α – лівий розмах нечіткого числа,

β – правий розмах нечіткого числа.

Триангулярне нечітке L-R число записується як $\widetilde{M}$ = (m, α, β)$_{LR.}$ Функція приналежності такого числа наведена на рисунку 5.2.

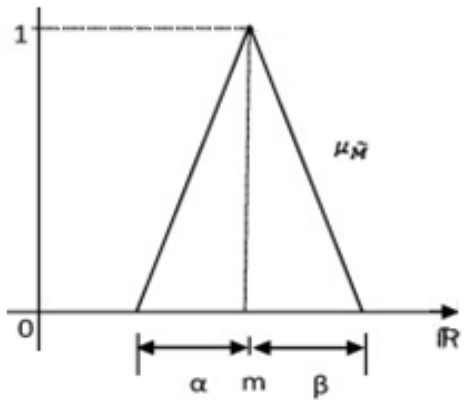

Рисунок 5.2 – Функція приналежності нечіткого триангулярного L-R – числа

Теорема 1. Два L-R нечітких числа $\widetilde{M} = (m, \alpha, \beta)_{LR}$ та $\widetilde{N} = (n, \gamma, \delta)_{LR}$ рівні тоді і тільки тоді, коли $m = n$, $\alpha = \gamma$, $\beta = \delta$ [15].

Теорема 2. Нехай $\widetilde{M} = (m, \alpha, \beta)_{LR}$ і $\widetilde{N} = (n, \gamma, \delta)_{LR}$ два нечіткі числа L-R – типу. Тоді справедливі рівності (5.2), (5.3), (5.4) [15].

$$(m, \alpha, \beta)_{LR} \oplus (n, \gamma, \delta)_{LR} = (m + n, \alpha + \gamma, \beta + \delta)_{LR}, \quad (5.2)$$

$$-(m, \alpha, \beta)_{LR} = (-m, \beta, \alpha)_{RL}, \quad (5.3)$$

$$(m, \alpha, \beta)_{LR} \ominus (n, \gamma, \delta)_{LR} = (m - n, \alpha + \delta, \beta + \gamma)_{LR}. \quad (5.4)$$

Теорема 3. Нечітке триангулярне число L-R типу $\widetilde{M} = (m, \alpha, \beta)_{LR}$ невід'ємне тоді і тільки тоді, коли $m \geq 0$, $m - \alpha \geq 0$, $m + \beta \geq 0$ [15].

Визначимо операції множення триангулярних нечітких чисел для двох операндів, що можуть набувати як додатних, так і від'ємних значень.

Визначимо операцію множення для додатних значень $\widetilde{M}$ та $\widetilde{N}$ (5.5) [15].

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} \approx (mn, m\gamma + n\alpha, m\delta + n\beta)_{LR}. \quad (5.5)$$

Визначимо операцію множення для додатного значення $\widetilde{N}$ та від'ємного значення $\widetilde{M}$ (5.6) [15].

$$(m, \alpha, \beta)_{RL} \otimes (n, \gamma, \delta)_{LR} \approx (mn, n\alpha - m\delta, n\beta - m\gamma)_{RL}. \quad (5.6)$$

Визначимо операцію множення для від'ємних значень $\widetilde{N}$ та $\widetilde{M}$ (5.7) [15].

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} \approx (mn, -n\beta - m\delta, -n\alpha - m\gamma)_{RL}. \quad (5.7)$$

Визначимо метод для порівняння двох нечітких триангулярних L-R чисел [14].

Теорема 4. Нехай $\widetilde{M} = (m_1, \alpha_1, \beta_1)_{LR}$ і $\widetilde{N} = (m_2, \alpha_2, \beta_2)_{LR}$ два нечіткі числа L-R – типу. Кажуть, що $\widetilde{M}$ відносно менше чим $\widetilde{N}$, що позначається як $\widetilde{M} < \widetilde{N}$ тоді і тільки тоді, коли виконується одна з умов (5.8), (5.9), (5.10) [15].

$$m_1 < m_2, \tag{5.8}$$

$$m_1 = m_2 \text{ і } (\alpha_1 + \beta_1) > (\alpha_2 + \beta_2), \tag{5.9}$$

$$m_1 = m_2 \text{ і } (\alpha_1 + \beta_1) = (\alpha_2 + \beta_2), \text{ і } (2m_1 - \alpha_1 + \beta_1) < (2m_2 - \alpha_2 + \beta_2). \tag{5.10}$$

Зауваження. Рввності $m_1 = m_2$ і $(\alpha_1 + \beta_1) = (\alpha_2 + \beta_2)$, і $(2m_1 - \alpha_1 + \beta_1) < (2m_2 - \alpha_2 + \beta_2)$ виконуються тоді і тільки тоді, коли $\widetilde{M} = \widetilde{N}$ [15].

Визначенн 3. Рангова функція – це функція $R : F(R) \rightarrow R$, де F(R) – набір нечітких чисел, визначених на множині реальних чисел, що ставлять у відображення кожному нечіткому числу – дійсне значення, де існує натуральний порядок.

Теорема 5. Нехай $\widetilde{M} = (m, \alpha, \beta)_{LR}$ нечітке число L-R – типу. Тоді рангова функція визначається за формулою (5.11).

$$R(\widetilde{M}) = m + (\beta - \alpha) / 4. \tag{5.11}$$

Зауваження. Якщо $\widetilde{M} = (a, b, c)$ – трикутне нечітке число в простому вигляді функція порядку визначається за формулою (5.12).

$$R(\widetilde{M}) = (a + 2b + c) / 4. \tag{5.12}$$

Трапезоїдні числа теж простіше представляти в L-R вигляді. Такі числа ще називають платоподібними.

Визначення 4. Трапезоїдне нечітке L-R число - це нечітке число M таке, що $\exists (m_1,m_2) \in R$, $m_1 < m_2$ і $\mu_m(x) = 1$ для будь-якого $x \in [m_1, m_2]$. Трапезоїдне нечітке число може моделювати нечіткий інтервал. Трапезоїдне нечітке число L-R типу M визначається з системи (5.13) [15].

$$\mu_m(x) = \begin{cases} L((m_1 - x)/\alpha) & x \le m_1,\ \alpha > 0 \\ R((x - m_2)/\beta) & x \ge m_2,\ \beta > 0 \\ 1 & \text{в іншому випадку} \end{cases}. \quad (5.13)$$

Функція приналежності нечіткого трапезоїдного LR – числа наведена на рисунку 5.3.

Більш коротке позначення для числа M, де L та R функції належності має вигляд: $(m_1, m_2, \alpha, \beta)_{LR}$.

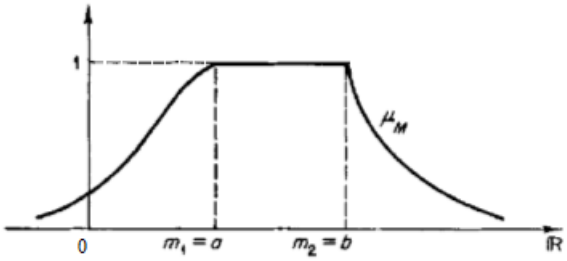

Рисунок 5.3 – Функція приналежності нечіткого трапезоїдного LR – числа

Розширені операції для значень $M > 0$ та $N > 0$ наедені в рівностях (5.14), (5.15) [15].

$$(m_1, m_2, \alpha, \beta)_{LR} \oplus (n_1, n_2, \gamma, \delta)_{LR} = \\ = (m_1 + n_1, m_2 + n_2, \alpha + \gamma, \beta + \delta)_{LR}, \quad (5.14)$$

$$(m_1, m_2, \alpha, \beta)_{LR} \odot (n_1, n_2, \gamma, \delta)_{LR} = \\ = (m_1 n_1, m_2 n_2, m_1\gamma + n_1\alpha,\ m_2\delta + n_2\beta)_{LR}. \quad (5.15)$$

Дослідимо операції над деякими типами нечітких чисел в середовищі Matlab. Результати досліджень наведені на рисунках 5.4, 5.5, 5.6.

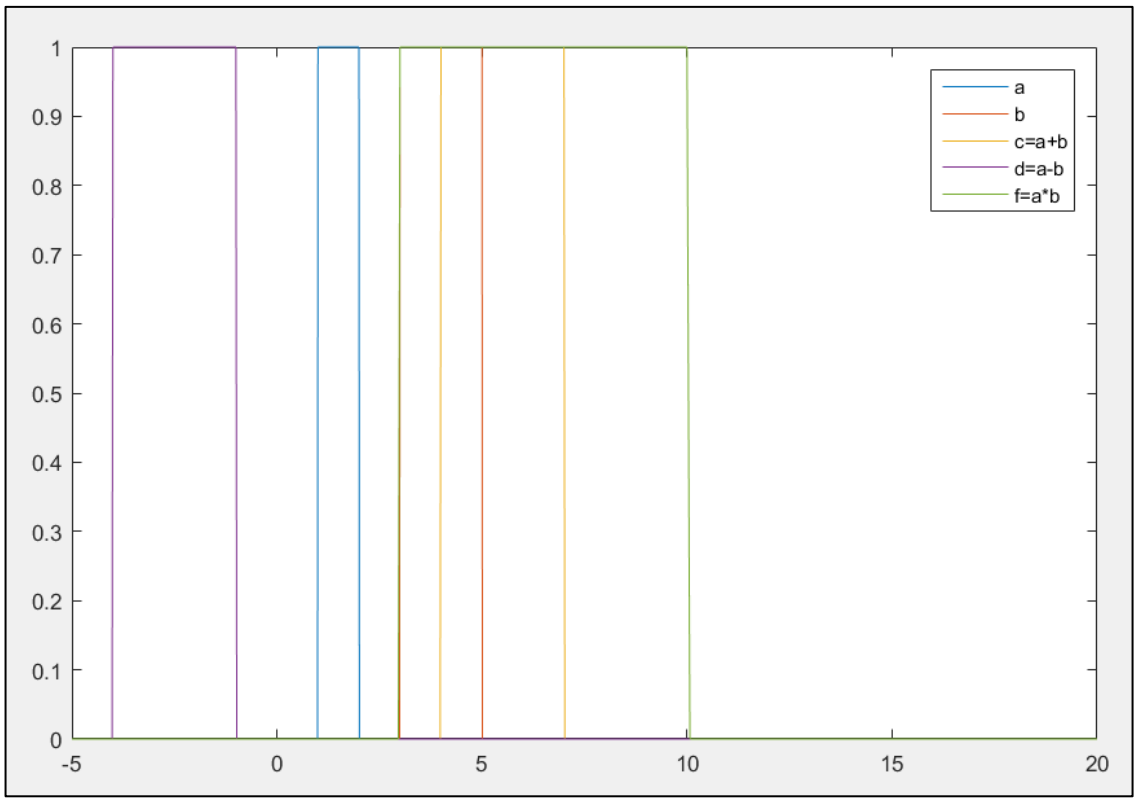

Рисунок 5.4 – Дослідження операцій суми, різниці та добутку над інтервальними нечіткими числами в середовищі Matlab

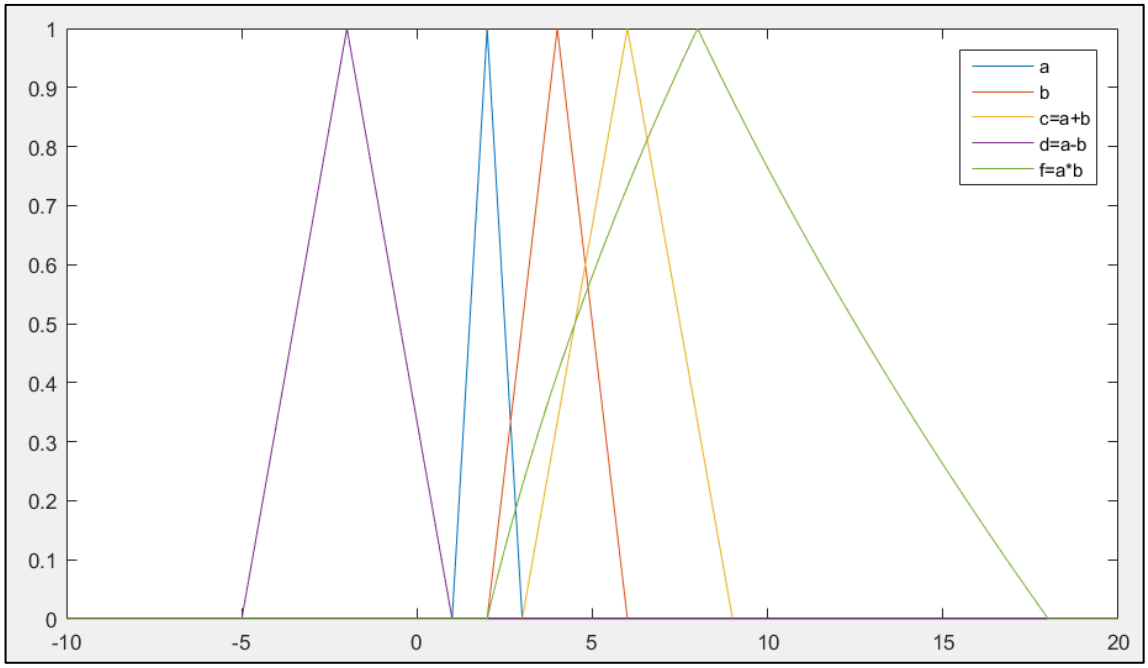

Рисунок 5.5 – Дослідження операцій суми, різниці та добутку над триангулярними нечіткими числами в середовищі Matlab

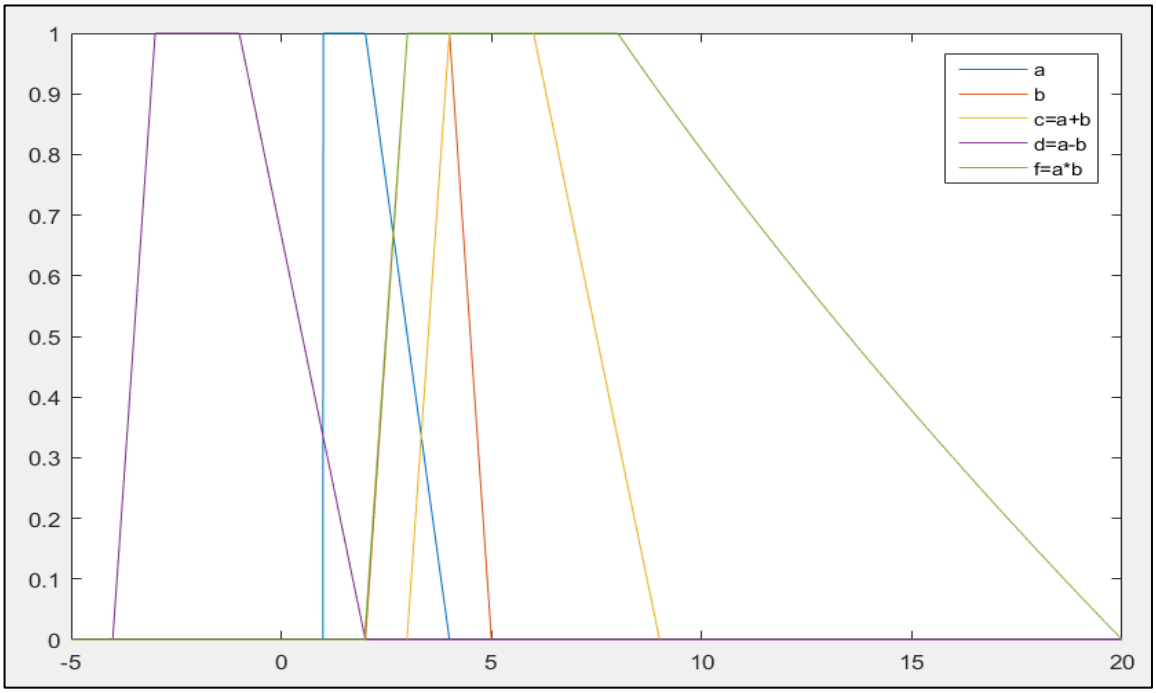

Рисунок 5.6 – Дослідження операцій суми, різниці та добутку над трапезоїдними нечіткими числами в середовищі Matlab.

Типи нечітких чисел, отриманих в результаті виконання арифметичних операцій над ними наведені в таблиці 5.1.

Таблиця 5.1 Типи нечітких чисел отриманих в результаті виконання трьох арифметичних операцій.

Операнди		Операції над нечіткими числами		
Операнд № 1	Операнд № 2	Сума	Різниця	Добуток
Чітке	Інтервальне	Інтервальне	Інтервальне	Інтервальне
Чітке	Триангулярне	Триангулярне	Триангулярне	Триангулярне
Чітке	Трапезоїдне	Трапезоїдне	Трапезоїдне	Трапезоїдне
Інтервальне	Інтервальне	Інтервальне	Інтервальне	Інтервальне
Інтервальне	Триангулярне	Трапезоїдне	Трапезоїдне	Трапезоїдне
Інтервальне	Трапезоїдне	Трапезоїдне	Трапезоїдне	Трапезоїдне
Триангулярне	Триангулярне	Трапезоїдне	Трапезоїдне	Триангулярне (з нелінійними сторонами)

Триангулярне	Трапезоїдне	Трапезоїдне	Трапезоїдне	Трапезоїдне (з нелінійними сторонами)
Трапезоїдне	Трапезоїдне	Трапезоїдне	Трапезоїдне	Трапезоїдне (з нелінійними сторонами)

5.2 Задача нечіткого однокритеріального програмування

Алгоритми нечіткого лінійного програмування (НЛП) дозволяють працювати з неточностями як в параметрах цільових функцій, так і в обмеженнях, що дає змогу реалізувати більш реалістичні моделі.

Нечітке програмування – це різновид математичного програмування, де використовуються нечіткі змінні. Нехай, x – це вектор рішень, а ξ – вектор з нечіткими параметрам. Так як нечітка цільова функція f(x, ξ) не може бути мінімізована напряму, ми можемо мінімізувати її очікуване значення (5.16)[16].

$$\min_x E[f(x,\xi)]. \tag{5.16}$$

Крім того, для обмежень $g_j(x,\ \xi) \le 0, j = 1, 2, \dots, p$ необхідно ввести рівні довіри α_1, α_2,…,α_p, оскільки кожне обмеження може мати різний вплив. Отже ми маємо набір випадкових обмежень з певним рівнем довіри (5.17).

$$M\{g_j(x,\xi) \le 0\} \ge \alpha_j, \text{ j} = 1, 2, \dots, \text{p}. \tag{5.17}$$

Для того, щоб отримати рішення з мінімально очікуваним значенням цільової функції по відношенню до випадкових обмежень використовується модель нечіткого програмування (5.18).

$$\begin{cases} \min\limits_{x} E[f(x,\xi)] \\ \text{по відношенню до:} \\ \qquad M\{g_j(x,\xi) \le 0\} \ge \alpha_j, \ j = 1,2,\dots,p. \end{cases} \tag{5.18}$$

Введемо наступні визначення з метою отримати метод розв'язку задачі нечіткого однокритеріального програмування.

Визначення 3. 1. Вектор x називається можливим рішенням то моделі нечіткого програмування, якщо виконується умова (5.19).

$$M\{g_j(x,\xi) \le 0\} \ge \alpha_j, j = 1,2,\dots,p. \tag{5.19}$$

Визначення 3. 2. Можливе рішення x^* називається оптимальним рішенням до моделі нечіткого програмування, якщо виконується умова (5.20)

$$E[f(x^*,\xi)] \le E[f(x,\xi)]\,. \tag{5.20}$$

Це має бути для будь-якого можливого рішення x.

5.3 Задача нечіткого багатокритеріального програмування

Було відмічено той факт, що більшість реальних проблем з прийняття рішень включають в себе декілька, неспіврозмірних та конфліктуючих цілей, що повинні розглядатися одночасно. Для того щоб оптимізувати одночасно декілька цілей, використовується багатокритеріальне програмування, що нині поширене дуже широко. Для моделювання багатокритеріальних задач прийняття рішень з невизначеними параметрами застосовується модель невизначеного багатокритеріального програмування (5.21) [16].

$$\begin{cases} \min\limits_{x}(E[f_1(x,\xi)], E[f_2(x,\xi)], \dots, E[f_m(x,\xi)]) \\ \text{по відношенню до:} \\ \quad M\{g_j(x,\xi) \le 0\} \ge \alpha_j, \ j = 1,2,\dots,p. \end{cases} \quad (5.21)$$

де

$f_i(x, \xi)$ – цільові функції при i = 1,2,…,m,

$g_j(x, \xi)$ – функції обмежень,

α_j – рівні довіри для j = 1,2,…,p.

Так як цілі зазвичай конфліктують між собой, немає оптимального рішення, що одночасно мінімізує всі цільові функції. В даному випадку, ми маємо ввести поняття рішення Паретто. Воно означає, що неможливо покращити яку небуть одну ціль, не жертвуючи одною чи декількома іншими цілями.

Визначення. Можливе рішення x* вважається Паретто по відношенню до багатокритеріального програмування, якщо не існує дупустимого рішення x такого, виконуються умови (5.22), (5.23) для хоча б одного індексу j.

$$E[f_i(x,\xi)] \le E[f_i(x^*,\xi)], \ i = 1, 2, \dots, m. \quad (5.22)$$

$$E[f_i(x,\xi)] < E[f_i(x^*,\xi)], \ i = 1, 2, \dots, m. \quad (5.23)$$

Якщо у суб’єкта, що приймає рішення, є функція переваги, що агрегує m цільових функцій, то ми можемо мінімізувати агрегуючу функцію переваг за умови того ж набору випадкових обмежень. Ця модель називається компромісною а, рішення називається компромісним рішенням. Було доведено, що компромісне рішення є Парето оптимальним відносно оригінальної багатокритеріальної моделі.

Перша добре-відома компромісна модель – це встановлення ваг для цільових функцій (5.24).

$$\begin{cases} \min\limits_{x} \sum\limits_{i=1}^{m} \lambda_i E[f_i(x,\xi)] \\ \text{по відношенню до:} \\ M\{g_j(x,\xi) \le 0\} \ge \alpha_j, \ j = 1,2,\dots,p. \end{cases} \quad (5.24)$$

де ваги $\lambda_1, \lambda_{2,\dots,}\lambda_m$ – невід'ємні числа, $\lambda_1 + \lambda_2 + \dots + \lambda_m = 1$, наприклад, $\lambda_i = 1/m$ для i = 1,2,…,m.

Другий спосіб пов'язаний з мінімізацією функції відстані з рішення у формулі (5.25)

$$(E[f_1(x,\xi)], E[f_2(x,\xi)], \dots, E[f_m(x,\xi)]). \quad (5.25)$$

до ідеального вектору $(f_1^*, f_2^*, \dots, f_m^*)$, f_1^*, де f_i^* - оптимальні значення i-х цільових функцій без врахування інших цілей, I = 1,2,…,m, відповідно. Це формула (5.26)

$$\begin{cases} \min\limits_{x} \sum\limits_{i=1}^{m} \lambda_i (E[f_i(x,\xi)] - f_i^*)^2 \\ \text{по відношенню до:} \\ M\{g_j(x,\xi) \le 0\} \ge \alpha_j, \ j = 1,2,\dots,p, \end{cases} \quad (5.26)$$

де ваги $\lambda_1, \lambda_{2,\dots,}\lambda_m$ – невід'ємні числа, $\lambda_1 + \lambda_2 + \dots + \lambda_m = 1$, наприклад, $\lambda_i = 1/m$ для i = 1,2,…,m.

В третьому випадку компромісне рішення може бути знайдено через інтерактивний підхід, що складається з послідовності етапів прийняття рішень і етапів обчислень. Його суть полягає в тому, що процес прийняття рішень при багатокритеріальній оптимізації проходить за участю експерта – певної людини, кваліфікованої в певній предметній області. Сам процес рішення складається з двох етапів: знаходженне одного з можливих оптимальних рішень. Оцінка експертом даного рішення. Тобто задача ПЗ надати експерту простір найоптимальніших

рішень для декількох критеріїв. Задача вибору рішення лежить на експерті. При цьому в процесі прийняття рішень може брати участь декілька експертів.

Якщо одні цільові функції мають пріорітет перед іншими, критерій оптимальності можна визначити за лексикографічним порядком.

Головна особливість рішень по лексикографічному порядку полягая в тому, що існує вибір між критеріями. Лексикографічна впорядкованість вимагає рангування критеріїв, оскільки оптимізація певного критерію можлива лише тоді, коли був досягнутий оптимум для попередніх критеріїв. Це означає, що перший критерій має найбільший пріорітет, і тільки у випадку існування декількох рішень по даному критерію буде виконуватися пошук рішень по другому та решті критеріїв.

Існування ієрархії між критеріями дає змогу вирішувати лексикографічні задачі послідовно, крок за кроком мінімізуючи по кожному наступному критерію, та використовуючи оптимальні значення попередніх критеріїв як обмеження.

5.4 Задача нечіткого багаторівневого програмування

Багаторівневе програмування пропонує засоби для вивчення децентралізованих систем прийняття рішень, в яких ми допускаємо, що є головна система, яка приймає рішення та підсистеми, які можуть мати свої власні змінні рішень та цільові функції. Головна система може тільки впливати на реакції підсистем через свої власні змінні рішення в той час як підсистеми мають повне право вирішувати, як оптимізувати свої власні цільові функції в контексті рішень головної системи та інших підсистем [16].

Розглянемо для прикладу дворівневу систему прийняття рішень. В дворівневій децентралізованій системі прийняття рішень є одна головна система та m підсистем. Нехай x та y_i будуть векторами рішень головної системи та i-ї підсистеми, i = 1, 2, … , m, відповідно. Ми також допускаємо, що цільові функції головної системи та i-ї підсистеми є: $F(x, y_1, \dots, y_m, \xi)$ та $f_i(x, y_1, \dots, y_m, \xi)$, i = 1, 2, … ,m, відповідно, де ξ - нечіткий вектор.

Нехай допустимий набір керуючого вектора x головної системи визначається випадковим обмеженням (5.27).

$$M\{G(x,\xi) \leq 0\} \geq \alpha, \qquad (5.27)$$

де

G – це функція обмеження,

α – встановлений рівень довіри.

Потім, для кожного рішення x, обраного головною системою виконуваність контрольних векторів y_i, i – их підсистем повинна бути залежна не тільки від x, а й також від $y_1,\ldots,y_{i-1},\ldots,y_{i+1},\ldots,y_m$ і загалом представлена випадковими обмеженням (5.28).

$$M\{g_j(x,y_1,y_2,\ldots,y_m,\xi) \leq 0\} \geq \alpha_j, \; j = 1,2,\ldots,p. \qquad (5.28)$$

де

g_i – функції обмежень,

α_i – наперед визначені рівні довіри, $i = 1, 2, \ldots, m$.

Припустимо, що головна система спочатку обирає власний контрольний вектор x, і підсистеми визначають свої контрольні масиви (y_1, y_2, …, y_m) після цього.

Щоб звести до мінімуму очікувану ціль головної системи використовується нечіке багаторівневе програмування (5.29).

$$\begin{cases} \min\limits_{x} E[F(x, y_1^*, y_2^*, \dots, y_m^*, \xi)] \\ \text{по відношенню до:} \\ \quad M\{G(x,\xi) \le 0\} \ge \alpha \\ \quad (y_1^*, y_2^*, \dots, y_m^*) \text{ вирішує проблеми } (i = 1,2,\dots,m) \\ \quad \begin{cases} \min\limits_{y_1} E[f_i(x, y_1, y_2, \dots, y_m, \xi)] \\ \quad \text{по відношенню до:} \\ \quad M\{g_j(x, y_1, y_2, \dots, y_m, \xi) \le 0\} \ge \alpha_i \end{cases} \end{cases} \quad (5.29)$$

5.5 Лексикографічний метод вирішення задач нечіткого програмування

Чисельний метод, в базовому вигляді ми використати не можемо, оскільки він призначений для звичайних («чітких») чисел. Але ми можемо звести задачу нечіткого лінійного програмування до чисельного методу використовуючи лексикографічний метод [17]. Суть даного методу полягає в тому, що цільова функція з нечіткими параметрами зводиться до набору цільових функцій з чіткими параметрами, що задовольняються послідовно одна за одною, а набір обмежень з нечіткими параметрами зводиться но набору обмежень з чіткими.

Якщо цільові функції та функції обмежень є монотонними по відношенню до нечітких параметрів (не змінних), то нечітка модель програмування може бути зведена то чіткої моделі математичного програмування. На практиці, більшість цільових та обмежуючих функцій є монотонними. З математичної точки зору, різниці між чітким математичним програмуванням та класичним математичним програмуванням немає, за вийнятком операції інтегрування. Тому дану задачу можна вирішувати різними методами математичної оптимізації. В даній роботі був обраний симплекс-метод**.**

Нехай маємо нечітку цільову функцію (5.30).

$$\max \text{ (or } \min) \ \tilde{C}^t \otimes \tilde{x}, \quad (5.30)$$

та обмеження (5.31)

$$\tilde{A} \otimes \tilde{x} = \tilde{b}, \tag{5.31}$$

де

$\tilde{C}^t$ – вектор нечітких коефіцієнтів при цільовій функції,

$\tilde{x}$ – вектор невід'ємних нечітких змінних,

$\tilde{A}$ – матриця нечітких коефіцієнтів при змінних в обмеженнях,

$\tilde{b}$ – вектор коефіцієнтів в обмеженнях.

Отже, $\tilde{C}^t = [\tilde{c}_j]_{1\,x\,n}$, $\tilde{A} = [\tilde{a}_{ij}]_{m\,x\,n}$, $\tilde{x} = [\tilde{x}_j]_{n\,x\,1}$, $\tilde{b} = [\tilde{b}_i]_{m\,x\,1}$, де $\tilde{c}_j$, $\tilde{X}_j$, $\tilde{a}_{ij}, \tilde{b}_i \in F(R)$ нечіткі числа L-R типу. Далі запропонуємо наступний метод розв'язання даної задачі.

Нехай виконуються рівності (5.32) – (5.36).

$$\tilde{C}^t\tilde{x} = ((c^t x)^m, (c^t x)^l, (c^t x)^u)_{LR}, \tag{5.32}$$

$$\tilde{A}\tilde{x} = ((\,Ax)^m, (Ax)^l, (Ax)^u)_{LR}, \tag{5.33}$$

$$\tilde{b} = ((b)^m, (b)^l, (b)^u)_{LR}, \tag{5.34}$$

$$\tilde{x} = ((x)^m, (x)^l, (x)^u)_{LR}, \tag{5.35}$$

тоді опишемо наступні кроки для вирішення даної задачі.

Відповідно до умови задачі маємо цільову функцію (5.36)

$$\max(min) \; = ((c^T x)^m, (c^T x)^l, (c^T x)^u)_{LR}, \tag{5.36}$$

при обмеженнях (5.37) – (5.40)

$$((Ax)^m, (Ax)^l, (Ax)^u)_{LR} = ((b)^m, (b)^l, (b)^u)_{LR}. \quad (5.37)$$

$$(x)^m \geq 0. \quad (5.38)$$

$$(x)^m - (x)^l \geq 0. \quad (5.39)$$

$$(x)^m + (x)^u \geq 0. \quad (5.40)$$

Крок 1. Перейдемо до наступної моделі з цільовою функцією (5.41)

$$\max(min) = ((c^T x)^m, (c^T x)^l, (c^T x)^u)_{LR}, \quad (5.41)$$

при обмеженнях (5.42) – (5.47).

$$(Ax)^m = (b)^m, \quad (5.42)$$

$$(Ax)^l = (b)^l, \quad (5.43)$$

$$(Ax)^u = (b)^u, \quad (5.44)$$

$$(x)^m \geq 0, \quad (5.45)$$

$$(x)^m - (x)^l \geq 0, \quad (5.46)$$

$$(x)^m + (x)^u \geq 0. \quad (5.47)$$

Крок 2. Виконаємо зведення нечіткої цільвої функції до трьох цільових функцій з чіткими параметрами (5.48) – (5.50).

$$\max(min) \ (c^T x)^m, \quad (5.48)$$

$$\min(max) \ (c^T x)^u + (c^T x)^l, \quad (5.49)$$

$$\max(min) \ 2(c^T x)^m - (c^T x)^l + (c^T x)^u, \quad (5.50)$$

при обмеженнях (5.51) – (5.56).

$$(Ax)^m = (b)^m, \quad (5.51)$$

$$(Ax)^l = (b)^l, \quad (5.52)$$

$$(Ax)^u = (b)^u, \quad (5.53)$$

$$(x)^m \geq 0, \quad (5.54)$$

$$(x)^m - (x)^l \geq 0, \quad (5.55)$$

$$(x)^m + (x)^u \geq 0. \quad (5.56)$$

Крок 3. Вирішується задача лінійного програмування з врахуванням першої цільової функції без врахування інших цільових функцій (5.57).

$$\max(min) \ (c^T x)^m, \quad (5.57)$$

при обмеженнях (5.58) – (5.63).

$$(Ax)^m = (b)^m, \quad (5.58)$$

$$(Ax)^l = (b)^l, \quad (5.59)$$

$$(Ax)^u = (b)^u, \quad (5.60)$$

$$(x)^m \geq 0, \quad (5.61)$$

$$(x)^m - (x)^l \geq 0, \quad (5.62)$$

$$(x)^m + (x)^u \geq 0. \quad (5.63)$$

Якщо дана задача має оптимальне рішення $\tilde{x}^* = ((x^*)^m, (x^*)^l, (x^*)^u)_{LR}$, то воно є оптимальним і ми зупиняємо алгоритм. Інакше переходимо до наступного кроку.

Крок 4. Вирішується задача лінійного програмування з врахуванням другої цільової функції без врахування інших цільових функцій (5.64).

$$\min(max) \ (c^T x)^u + (c^T x)^l, \quad (5.64)$$

при обмеженнях (5.64) – (5.70).

$$(c^T x)^m = s^* \quad (5.64)$$

$$(Ax)^m = (b)^m, \quad (5.65)$$

$$(Ax)^l = (b)^l, \quad (5.66)$$

$$(Ax)^u = (b)^u , \quad (5.67)$$

$$(x)^m \geq 0, \quad (5.68)$$

$$(x)^m - (x)^l \geq 0, \quad (5.69)$$

$$(x)^m + (x)^u \geq 0. \quad (5.70)$$

Якщо дана задача має оптимальне рішення $\tilde{x}^* = ((x^*)^m, (x^*)^l, (x^*)^u)_{LR}$, то воно є оптимальним і ми зупиняємо алгоритм. Інакше переходимо до наступного кроку.

Крок 5. Вирішується задача лінійного програмування з врахуванням третьої цільової функції без врахування інших цільових функцій (5.71).

$$2(c^T x)^m - (c^T x)^l + (c^T x)^u, \quad (5.71)$$

при обмеженнях (5.72) - (5.79).

$$(c^T x)^u + (c^T x)^l = n^*, \quad (5.72)$$

$$(c^T x)^m = s^*, \quad (5.73)$$

$$(Ax)^m = (b)^m, \quad (5.74)$$

$$(Ax)^l = (b)^l, \quad (5.75)$$

$$(Ax)^u = (b)^u , \quad (5.76)$$

$$(x)^m \geq 0, \quad (5.77)$$

$$(x)^m - (x)^l \geq 0 , \quad (5.78)$$

$$(x)^m + (x)^u \geq 0. \quad (5.79)$$

де n^* - оптимальне значення на кроці (4). Отже, якщо дана задача має оптимальне рішення $\tilde{x}^* = ((x^*)^m, (x^*)^l, (x^*)^u)_{LR}$,то воно є оптимальним і ми отримуємо остаточне рішення. Інакше задача рішення не має.

5.6 Використання генетичного алгоритму для багаторівневої задачі

В задачі розподілу ресурсів критичної IT-інфраструктури постає проблема оптимального розподілу ресурсів за декількома критеріями оптимальності на декілька рівнях. Підхід до вирішення даної проблеми не є тривіальним. Оскільки має обслуговуватись максимальна кількість бізнес-процесів та універсальних сервісів, то необхідно забезпечити їх оптимальний розподіл між віртуальними машинами, а також, оптимальний розподіл віртуальних машини між серверами. Це задача повного перебору, тому доцільним рішенням буде застосування генетичного алгоритму.

Генетичний алгоритм – це алгоритм пошуку, заснований на еволюційному підході, який використовується для вирішення задач оптимізації і моделювання методом послідовного підбору, комбінування і варіації параметрів. Даний алгоритм використовує механізми, схожі з біологічною еволюцією. Найважливіша частина генетичного алогоритму – використання оператора для «схрещення», що на кожному етапі еволюції «перетасовує» простір можливих рішень для нашої задачі.

Для того, щоб використати генетичний алгоритм, необхідно закодувати вихідну задачу так, щоб її розв’язок можна було б отримати у вигляді масиву даних. В даному випадку масив представляє собою «аналог» біологічної хромосоми. При кожній ітерації змінюється вміст «хромосоми» відповідно до типу функції мутації. Для того, щоб оцінити якість популяції використовується функція пристосування.

Схема роботи генетичного алгоритму.

Задача кодується таким чином, щоб її вирішення могло бути представлено в вигляді масиву подібного до інформації складу хромосоми. Цей масив часто називають саме так – «хромосома». Випадковим чином в масиві створюється деяка

кількість початкових елементів «осіб», або початкова популяція. Особи оцінюються з використанням функції пристосування, в результаті якої кожній особі присвоюється певне значення пристосованості, яке визначає можливість виживання особи. Після цього з використанням отриманих значень пристосованості вибираються особи, допущені до схрещення (селекція). До осіб застосовується «генетичні оператори» (в більшості випадків це оператор схрещення (crossover) і оператор мутації (mutation)), створюючи таким чином наступне покоління осіб. Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і мутація. Так моделюється еволюційний процес, що продовжується декілька життєвих циклів (поколінь), поки не буде виконано критерій зупинки алгоритму. Таким критерієм може бути:

- знаходження глобального, або надоптимального вирішення;
- вичерпання числа поколінь, що відпущені на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

Генетичні алгоритми можуть використати для пошуку рішень в дуже великих і важких просторах пошуку.

Етапи генетичного алгоритму

Можна виділити такі етапи генетичного алгоритму:

- Створення початкової популяції:
- Обчислення функції пристосованості для осіб популяції (оцінювання)
- Повторювання до виконання критерію зупинки алгоритму:
    1) Вибір індивідів із поточної популяції (селекція)
    2) Схрещення або/та мутація
    3) Обчислення функції пристосовуваності для всіх осіб
    4) Формування нового покоління

Створення початкової популяції.

Перед першим кроком необхідно випадковим чином створити деяку початкову популяцію. Навіть якщо популяція виявиться абсолютно неконкурентоздатною, генетичний алгоритм все одно достатньо швидко переведе

її в придатну для життя популяцію. Таким чином, на першому кроці можна не старатися зробити надто пристосованих осіб, достатньо, щоб вони відповідали формату осіб популяції, і на них можна було порахувати функцію пристосованості. Наслідком першого кроку є популяція H, що налічує N осіб. На етапі відбору необхідно із всієї популяції вибрати її певну долю, яка залишиться в «живих» на цьому етапі популяції. Є декілька способів провести відбір. Ймовірність виживання особи h повинна залежати від значення її пристосованості Fitness(h). Сама ж доля відібраних s зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. Внаслідок відбору із N осіб популяції H повинні залишитись sN осіб, які ввійдуть в наступну популяцію H'. Решта осіб «загине».

Розмноження.

Розмноження в різних алгоритмах описується по різному — воно, звісно, залежить від формату осіб. Головна вимога до розмноження — щоб нащадок чи нащадки мали можливість успадкувати риси всіх батьків, «змішавши» їх якимось достатньо розумним чином.

Розмноження або оператор рекомбінації застосовують відразу ж після оператора відбору батьків для отримання нових особин-нащадків. Сенс рекомбінації полягає в тому, що створені нащадки повинні наслідувати генну інформацію від обох батьків. Розрізняють дискретну рекомбінацію і кросинговер.

Особи для розмноження зазвичай вибираються із всієї популяції H, а не із тих, що вижили на першому кроці (хоча останній варіант теж має право на існування). Справа в тому, що головна проблема генетичних алгоритмів — недостача різноманітності (diversity) в особах. Достатньо швидко виділяється єдиний генотип, який являє собою локальний максимум і згодом всі елементи популяції програють йому в відборі, і вся популяція «забивається» копіями цієї особи. Існують різні способи боротьби із таким небажаним ефектом; один з них — вибір для розмноження не з самих «пристосованих», а взагалі зі всіх осіб.

Мутації.

До мутацій відноситься все те ж, що і до розмноження: є деяка доля мутантів m, що є параметром генетичного алгоритму, і на кроці мутацій необхідно вибрати mN осіб, а згодом змінити їх згідно з заздалегідь заданими операціями мутації.

5.7 Розробка моделі розподілу ресурсів критичної IT-інфраструктури

Припустимо, що для надання послуги хмарних обчислень було виділено S_i, $i = 1..n$ фізичних серверів, на яких під управлінням гіпервізорів працюють віртуальні машини (ВМ) V_i, $i = 1..m$.

Далі введемо необхідні позначення для формування моделі розподілу ресурсів критичної IT-інфраструктури:

$Z_1, \dots, Z_m$– комплекс бізнес-процесів, підтримка яких забезпечує ефективне функціонування об'єкта управління;

$S_1, \dots, S_m$ – комплекс універсальних сервісів, підтримка яких забезпечує ефективне функціонування об'єкта управління;

$w_i^z, \dots, w_n^z$– коефіцієнти критичності бізнес-процесів $Z_1, \dots, Z_n$ відповідно за нечіткою шкалою:

- критичний - $\alpha_1 \le w_i^z \le 1$;
- дуже важливий -$\alpha_2 \le w_i^z \le \alpha_1$;
- важливий -$\alpha_3 \le w_i^z \le \alpha_2$;
- не важливий - $w_i^z \le \alpha_3$.

$w_i^s, \dots, w_m^s$– коефіцієнти критичності універсальних сервісів $S_1, \dots, S_n$ відповідно за нечіткою шкалою:

- критичний - $\gamma_1 \le w_i^s \le 1$;
- дуже важливий -$\gamma_2 \le w_i^s \le \gamma_1$;
- важливий -$\gamma_3 \le w_i^s \le \gamma_2$;
- не важливий - $w_i^s \le \gamma_3$.

$R_1, \dots, R_m$ – ресурси сервера S_i критичної IT-інфраструктури у нодах, що необхідні для підтримки бізнес-процесів;

$T_1, \dots, T_m$- надійність ресурсів сервера S_i критичної IT-інфраструктури;

$p_0 = \|\tilde{p}_{0ij}^k\|$ – матриця потреб бізнес-процесів у ресурсах критичної IT-інфраструктури на віртуальній машині V_k, яка гарантовано задовольняється, де $\tilde{p}_{0ij}^k$ дорівнює кількості потрібного для бізнес-процесу Z_i ресурсу R_j у вигляді тріангулярного нечіткого числа $\tilde{p}_{0ij}^k = (p_{0ij}^{k\text{г}},\ p_{0ij}^{k\text{с}}, p_{0ij}^{k\text{к}})$ чи 0, якщо ресурс не потрібен, де $p_{0ij}^{k\text{г}}$ - найгірший (максимальний) варіант потреби у ресурсі, $p_{0ij}^{k\text{с}}$ - середня потреба у ресурсі, $p_{0ij}^{k\text{к}}$ - найкращий (мінімальний) варіант потреби у ресурсі.

$s_0 = \|\tilde{s}_{0ij}^k\|$– матриця потреб універсальних сервісів у ресурсах критичної IT-інфраструктури на віртуальній машині V_k, яка гарантовано задовольняється, де $\tilde{s}_{0ij}^k$ дорівнює кількості потрібного для універсального сервісу S_i ресурсу R_j у вигляді тріангулярного нечіткого числа $\tilde{s}_{0ij}^k = (s_{0ij}^{k\text{г}},\ s_{0ij}^{k\text{с}}, s_{0ij}^{k\text{к}})$ чи 0, якщо ресурс не потрібен, де $s_{0ij}^{k\text{г}}$ - найгірший (максимальний) варіант потреби у ресурсі, $s_{0ij}^{k\text{с}}$ - середня потреба у ресурсі, $s_{0ij}^{k\text{к}}$ - найкращий (мінімальний) варіант потреби у ресурсі.

$p = \|\tilde{p}_{ij}^l\|$ – матриця потреб бізнес-процесів у ресурсах критичної IT-інфраструктури на віртуальній машині $V_l (l \neq k)$, які додатково бажано зарезервувати, де $\tilde{p}_{ij}^l$ дорівнює кількості потрібного для бізнес-процесу Z_i ресурсу R_j у вигляді тріангулярного нечіткого числа $\tilde{p}_{ij}^l = (p_{ij}^{l\text{г}},\ p_{ij}^{l\text{с}}, p_{ij}^{l\text{к}})$ чи 0, якщо ресурс не потрібен, де $p_{ij}^{l\text{г}}$ - найгірший (максимальний) варіант потреби у ресурсі, $p_{ij}^{l\text{с}}$ - середня потреба у ресурсі, $p_{ij}^{l\text{к}}$ - найкращий (мінімальний) варіант потреби у ресурсі.

$s = \|\tilde{s}_{ij}^l\|$ – матриця потреб універсальних сервісів у ресурсах критичної IT-інфраструктури на віртуальній машині $V_l (l \neq k)$, які додатково бажано зарезервувати, де $\tilde{s}_{ij}^l$ дорівнює кількості потрібного для універсального сервісу S_i ресурсу R_j у вигляді тріангулярного нечіткого числа $\tilde{s}_{ij}^l = (s_{ij}^{l\text{г}},\ s_{ij}^{l\text{с}}, s_{ij}^{l\text{к}})$ чи 0, якщо ресурс не потрібен, де $s_{ij}^{l\text{г}}$, - найгірший (максимальний) варіант потреби у ресурсі, $s_{ij}^{l\text{с}}$ - середня потреба у ресурсі, $s_{ij}^{l\text{к}}$ - найкращий (мінімальний) варіант потреби у ресурсі.

$t = \|\tilde{t}_j^i\|$ – матриця надійності ресурсів критичної IT-інфраструктури на віртуальній машині V_i, де $\tilde{t}_j^i$ дорівнює надійності ресурсу R_j у вигляді триангулярного нечіткого числа $\tilde{t}_j^i = (t_j^{i\text{г}}, t_j^{i\text{с}}, t_j^{i\text{к}})$ чи 0, якщо надійність ресурсу не важлива, де $t_j^{i\text{г}}$, - найгірший (мінімальний) варіант надійності ресурсу, $t_j^{i\text{с}}$ - середня надійність ресурсу, $t_j^{i\text{к}}$ - найкращий (максимальний) варіант надійності ресурсу.

Середня потреба у ресурсі та надійність визначаються за формулами (5.69 – 5.73):

$$p_{0ij}^{k\text{с}} = \frac{p_{0ij}^{k\text{г}} + \beta p_{0ij}^{k\text{нв}} + p_{0ij}^{k\text{к}}}{2 + \beta_0} \tag{5.69}$$

$$p_{ij}^{l\text{с}} = \frac{p_{ij}^{l\text{г}} + \beta p_{ij}^{l\text{нв}} + p_{ij}^{l\text{к}}}{2 + \beta} \tag{5.70}$$

$$s_{0ij}^{k\text{с}} = \frac{s_{0ij}^{k\text{г}} + \chi s_{0ij}^{k\text{нв}} + s_{0ij}^{k\text{к}}}{2 + \chi_0} \tag{5.71}$$

$$s_{ij}^{l\text{с}} = \frac{s_{ij}^{l\text{г}} + \chi s_{ij}^{l\text{нв}} + s_{ij}^{l\text{к}}}{2 + \chi} \tag{5.72}$$

$$t_j^{i\text{с}} = \frac{t_j^{i\text{г}} + \tau t_j^{i\text{нв}} + t_j^{i\text{к}}}{2 + \tau} \tag{5.73}$$

Де $p_{0ij}^{k\text{нв}}, p_{ij}^{l\text{нв}}, s_{0ij}^{k\text{нв}}, s_{ij}^{l\text{нв}}$ - найімовірніші варіанти потреб у ресурсі для бізнес-процесу та універсального сервісу,

$t_j^{i\text{нв}}$ - найімовірніше значення надійності ресурсу, $\beta, \beta_0, \chi, \chi_0, \tau$ - зважені параметри усереднення, які визначаються експериментально.

$\tilde{r}_1^k, \dots, \tilde{r}_m^k$– триангулярні нечіткі числа вигляду $(r_j^{k\text{г}}, r_j^{k\text{с}}, r_j^{k\text{к}})$, що визначають кількість ресурсів $R_1, \dots, R_m$ відповідно на віртуальній машині V_k.

a_{ij} - булева змінна, яка визначає, чи встановлена ВМ V_j на сервері S_i.

Матриця розподілу віртуальних машин між серверами наведена на рисунку 4.3.

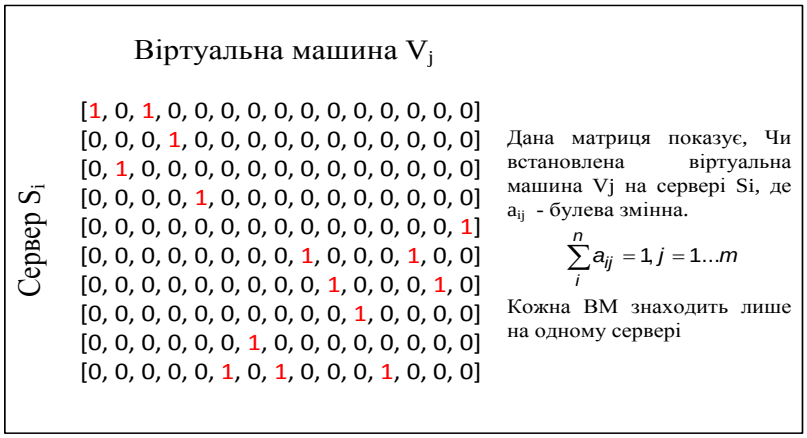

Рисунок 4.3 – Матриця відповідності розташування віртуальних машин на різних серверах

Очевидно, що для вказаної моделі розподілу ресурсів має виконуватись вимога $p_{ij}^l + p_{0ij}^k \geq p_{0ij}^k$.

У випадку, коли немає можливості для виділення додаткових ресурсів, попередня нерівність перетворюється у рівність $p_{ij}^l + p_{0ij}^k = p_{0ij}^k$.

Критичний процес або сервіс завжди обслуговується. Інші процеси або сервіси обслуговуються згідно коефіцієнтів критичності.

Введемо коефіцієнти при триангулярних нечітких змінних наступним чином:

$$x_i = \begin{cases} 1, & \text{якщо бізнес} - \text{процес } Z_1 \text{ є критичним і обслуговується в першу чергу} \\ \alpha_2, & \text{процес є дуже важливим і обслуговується} \\ \alpha_3, & \text{процес є важливим і обслуговується} \\ 0, & \text{процес не важливий і не обслуговується} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{якщо бізнес – процес } S_1 \text{ є критичним і обслуговується в першу чергу} \\ \gamma_2, & \text{процес є дуже важливим і обслуговується} \\ \gamma_3, & \text{процес є важливим і обслуговується} \\ 0, & \text{процес не важливий і не обслуговується} \end{cases}$$

Оскільки кожна ВМ розташована тільки на одному сервері, то має виконуватись наступна умова (5.74):

$$\sum_{i}^{n} a_{ij} = 1, j = 1..m \tag{5.74}$$

Умова критичності сервісів і процесів накладає наступне обмеження (5.75):

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}\, p_{0ij}^{kc} \le r_j^{k\text{r}}, k = 1..n \tag{5.75}$$

Таким чином, управління розподілом ресурсами критичної IT-інфраструктури зводиться до пошуку максимумів наступних цільових функцій одночасно (5.76):

$$\widetilde{U} = \sum_{i}^{n} \tilde{x}_i \tilde{w}_i^z + \sum_{j}^{m} \tilde{y}_j \tilde{w}_i^s \to max, \tag{5.76}$$

$$\widetilde{T} = \sum_{j}^{n} \tilde{t}_j^i \tilde{r}_j^k \to max,$$

за наявності обмежень (5.77) і (5.78):

$$\sum_{i=1}^{n} \tilde{x}_i \cdot \left(\tilde{p}_{0ij}^{k} + \tilde{p}_{ij}^{l}\right) + \sum_{i=1}^{m} \tilde{y}_i \cdot \left(\tilde{s}_{0ij}^{k} + \tilde{s}_{ij}^{l}\right) \leq \tilde{r}_j^k \quad (5.77)$$

$$N_{\text{кр}} \rightarrow max, \quad (5.78)$$

для $j = 1 \ldots m$ і $k, l = 1..n$, та врахуванням попередніх умов.

Умова визначає той факт, що кількість критичних процесів та універсальних сервісів, які необхідно обслужити є максимальною, тобто всі критичні процеси та сервіси повинні бути забезпечені ресурсами. В противному випадку управління розподілом ресурсів є неможливим і потрібно вводити додаткові ресурси.

Коефіцієнти $\widetilde{w}_i^z$, $\widetilde{w}_k^s$ цільової функції становлять собою нечіткі числа - $\widetilde{w}_i^z \in [w_i^{zL};\ w_i^{zR}]$, $\widetilde{w}_k^s \in [w_k^{sL};\ w_k^{sR}]$, L і R - ліві та праві границі носія нечіткого числа.

Задача становить собою задачу нечіткого лінійного програмування, яку можна розв’язати за допомогою методів, які описані в [11].

6. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Вибір технології та середовища розробки

Для розробки програмної бібліотеки з моделями та методами для розподілу ресурсів критичної IT-інфраструктури були обрані мови Python та C++.

Вибір мови Python обумовлений тим, що дана мова програмування є дуже зручна для створення різноманітних прототипів та моделей, а також інує велика кількість бібліотек для задач лінійного програмування, швидких обчислень, забезпечення прямого доступу до хмарних технологій з використанням певного клієнту. Також варто зазначити великі можливості Python щодо зручного представлення даних.

Динамічна типізація та зручний синтаксис, велика кількість вбудованих структур даних, а також простота створення нових структур даних – все це робить мову Python дуже зручною для швидкого прототипування, реалізації різних математичних моделей і т.п.

Для мови Python було обрано середовище розробки PyCharm.

PyCharm розроблений компанією JetBrains на основі IntelliJ IDEA. PyCharm являє собою інтегроване середовище розробки, що працює під операційними системам Windows, Mac OS X та Linus.

PyCharm наступні можливості користувачу з:

- налаштування програм;
- запуску юніт-тестів;
- підсвітки синтаксису та помилок;
- синтаксичного аналізу коду;
- навігації по проекту та вихідному коду програми;
- підтримки веб-розробки (фреймворк Django);
- відображенню файлової структури проекту;
- рефакторингу (перейменування змінних, введення змінної, констант, вилучення методів);

- підтримки систем контролю версій;

PyCharm – дуже зручне середовище розробки, оскільки воно надає широкі можливості для розробки основних компонентів програмного продукту, а також, великі можливості для тестування програмного забезпечення. Для того, щоб запустити процес виконання коду, необхідно також встановити інтерпретатор коду Python. В нашому випадку було обрано інтерпретатор Python 3.5.

Варто також розглянути один аспект. Оскільки мета даної дипломної роботи – створення програмної бібліотеки для ефективного використання ресурсів критичної IT-інфраструктури, і підхід до вирішення цієї задачі вимагає затратних алгоритмів.

Такі речі, як динамічна типізація, що використовується в Python можуть значно понизити швидкість виконання поставленої задачі. Тому було вирішено «важкі» ділянки коду реалізувати мовою C++ у вигляді бібліотеки, що потім дуже просто підключаєтсья до програми, написаної мовою Python.

Мова C++ чітко типізована, що дуже сильно зменшує ризики виникнення виключних ситуацій, та полегшує контроль над змінними програми.

Для розробки програм мовою C++ використовується середовище розробки Microsoft Visual Studio 2015.

Microsoft Visual Studio – це продук компанії Microsoft, що включає в себе інтегроване середовище розробки програмного забезечення та ряд інших інструментальних засобів. Дані продукти дать змогу розробляти як консольні застосуванна, так і застосування з графічним інтерфейсом, в тому числі і з підтримкою технології Windows Forms. Також Microsoft Visual Studio надає можливості для розробки веб-сайтів, веб-застосувань та веб-служб, для всіх платформ, котрі підтримують Windows Phone, Xbox, .Net Framework, Windows, Silverlight та ін.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense та можливістю найпростішого рефакторингу коду. Вбудуванний відладчик може працювати як відладчик рівня вихідного коду, так і як відладчик машинного рівня.

Visual Studio надає можливості підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи підтримку систем контролю версій вихідного коду, додавання нових інструментів, проектування коду на предметно-орієнтованих мовах програмування та ін.

Дане середовище розробки містить в будований компілятор для виконання програмного коду.

6.2 Структура розроблюваної бібліотеки

На даний момент бібліотека, розроблена на Python включає в себе набір модулів для реалізації необхідних структур даних, пошуку оптимальних рішень для задач лінійного програмування на простих числах, для задач нечіткого програмування з одним критерієм оптимальності, для задач нечіткого програмування з використанням лексикографічного методу, для багаторівневих задач нечіткого програмування, а також для моделювання структури самої хмари та процесу обслуговування бізнес-процесів і універсальних сервісів.

Для реалізації задачі нечіткого багаторівневого програмування використовеється генетичний алгоритм. Так як генетичний алгоритм потребує досить багато ресурсів, було вирішено винести реалізацію генетичного алгоритму в бібліотеку на мові C++, що інтегрується в проект на мові Python за допомогою пакету Boost.Python.

6.3 Зовнішні бібліотеки

Оскільки уже існують певні рішення щодо розв’язку окремих підзадач системи з розподілу ресурсів, використовуються деякі зовнішні бібліотеки.

Для вирішення задач нечіткого лінійного програмування з трапезоїдними числами існує бібліотека ROI на мові R, для якої розроблено обгортку на мові Python. Це дає змогу в подальшому порівнювати результати рішення розроблених

нами методів з уже існуючими рішеннями. Для інтеграції бібліотек на мовах R та Python використовується пакет rpy2.

Для вирішення простої задачі лінійного програмування використовується пакет lp_solve.

Для збільшення продуктивності виконання Python програм використовується пакет numpy.

6.4 Опис модулів розробленої бібліотеки

Дана бібліотека включає в себе декілька модулів для реалізації поставленого завдання:

- модуль fuzzynum.py для представлення нечітких чисел та операцій над ними;
- модуль crisp_lp.py для вирішення задачі лінійного програмування на простих числах;
- модуль обгортка general_lp.py для вирішення задачі нечіткого лінійного програмування на трапезоїдних числах;
- модуль lexicography_lp.py для реалізації лексикографічного методу вирішення задач нечіткого програмування, а також багатокритеріальних задач;
- модуль multilevel_solver.py для вирішення задачі багаторівневого нечіткого програмування.
- модуль cloud_solver.py для моделювання структури хмари та процесу обслуговування бізнес-процесів та універсальних сервісів.

Модуль fuzzy_num.py включає в себе реалізацію трьох типів нечітких чисел: інтервальних, триангулярних та трапезоїдних. Для всіх трьох типів нечітких чисел було перевизначено операції додавання, віднімання, множення, інверсії. Також було перевантажено операції порівняння нечітких чисел. Для кожного екземпляру нечіткого числа можна застосувати функцію впорядкування, що дає змогу звести нечітке число до скалярного значення.

Модуль crisp_lp.py являє собою обгортку для задачі лінійного програмування з простими числами. В даному модулі реалізовано зручний інтерфейс для вирішення задач лінійного програмування.

Модуль general_lp.py являє собою обгортку для бібліотеки ROI мовою R. В даному модулі реалізовано інтерфейс для виклику методів бібліотеки ROI, за допомогою простих команд Python.

Модуль lexicography_lp.py включає класи та методи для вирішення задачі нечіткого лінійного програмування з використанням лексикографічного програмування. Суть даного методу полягає в тому, що задача нечіткого лінійного програмування зводиться до задачі багатокритеріального програмування з простими числами.

Модуль multilevel_solver.py включає в себе класи та методи для вирішення багаторівневої задачі нечіткого лінійного програмування. Принцип роботи задачі нечіткого лінійного програмування наступний.

Модуль cloud_solver.py включає в себе набір класів та методів для моделювання структури хмари, а також процесу розподілу ресурсів між виконуваними одиницями (бізнес-процеси та універсальні сервіси).

7 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

7.1 Тестування модуля з класами нечітких чисел

Для реалізації класів з нечіткими числами було розроблено модуль fuzzynum.py на мові Python. В даному модулі реалізовані класи, методи та перевантаження операцій для трьох видів нечітких чисел: інтервальних, триангулярних, траезоїдних.

На рисунку 7.1 наведено тестування нечітких триангулярних чисел, а на рисунку 7.2 тестування нечітких трапезоїдних чисел.

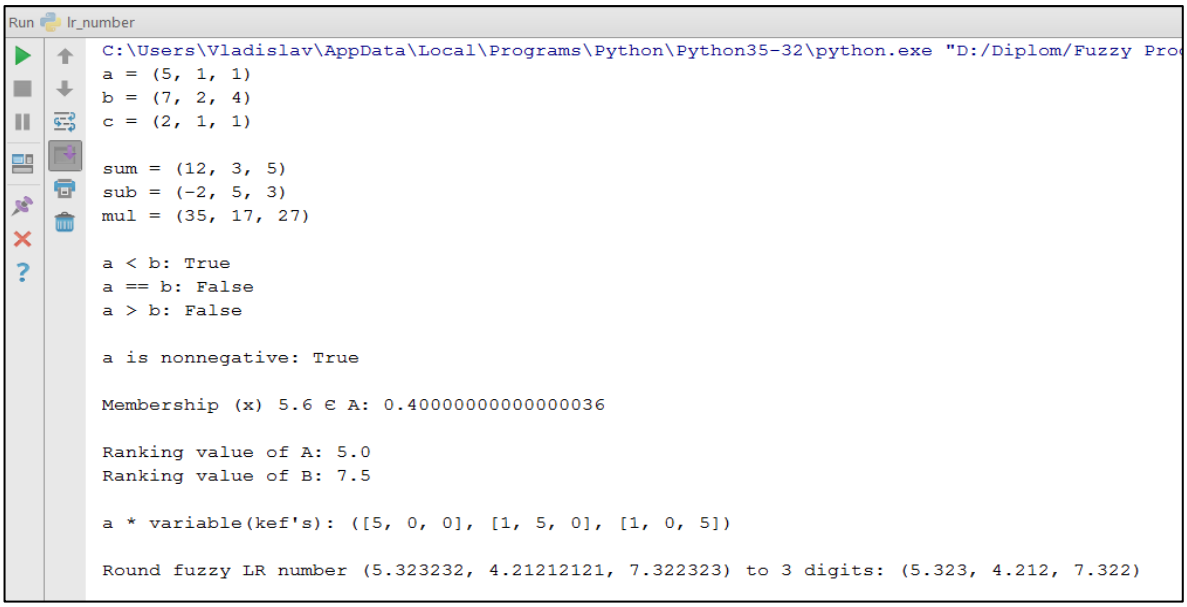

Рисунок 7.1 – Тестування класу з нечіткими триангулярними числами

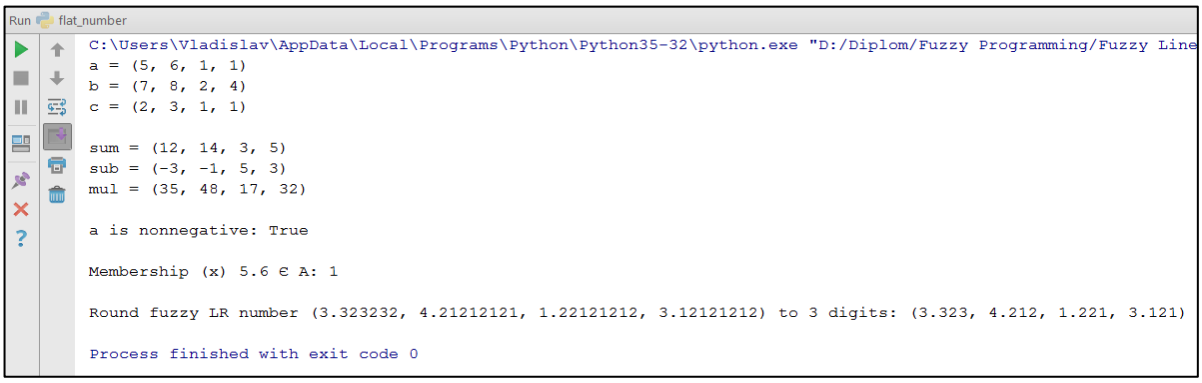

Рисунок 7.2 – Тестування класу з нечіткими триангулярними числами

7.2 Тестування модуля задачі лінійного програмування на звичайних числах

Для вирішення задачі лінійного програмування на звичайних числах використовується симплекс-метод. Для цього був розроблений модуль-обгортка simplex_solver.py на мові Python. Даний модуль використовує бібліотеки numpy, lpsolve55.

Задамо вхідні параметри та запустимо скрипт:

```
>>> from simplex_solver import *
>>> C = np.array([1, -2])
>>> A = np.array([[5, 3], [1, -1], [-3, 5]])
>>> b = np.array([30, 3, 15])
>>> solver = LPSolver()
>>> solver.solve("max", C, A, b, [">=", "<=", "<="])
```

Результат виконання:

[1.1250000000000009, [4.875000000000001, 1.875]],

де F = 1.125 – цільова функція, X = (4.875, 1.875) – вектор змінних.

Для іншого набору даних результат виконання наведено на рисунку 7.3.

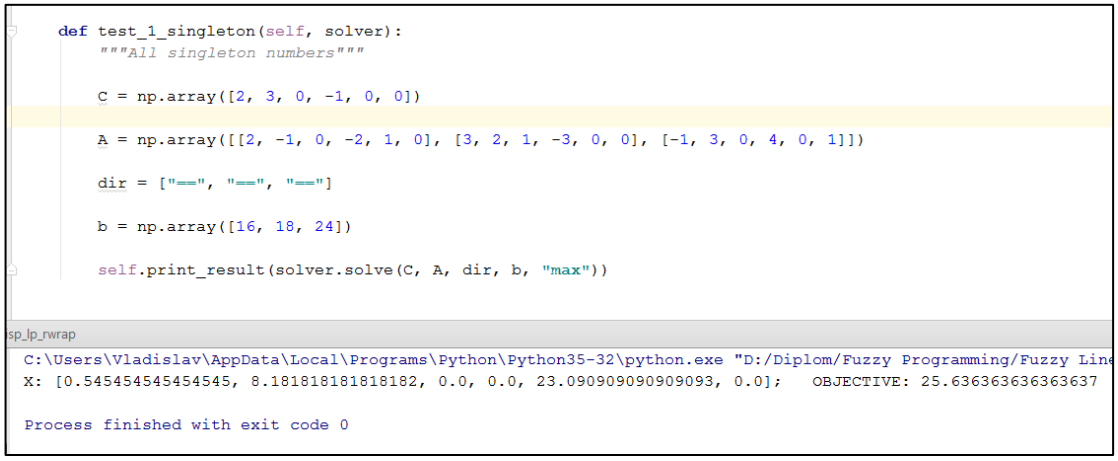

Рисунок 7.3 – Тестування задачі лінійного програмування на звичайних числах

7.3 Тестування модуля задачі нечіткого програмування

Для вирішення задачі нечіткого багатокритеріального програмування використовується лексикографічний метод. Застосування даного методу дало змогу звести задачу лінійного програмування на нечітких числах (LR типу) до звичайної задачі лінійного програмування.

Було розроблено модулі lr_number.py (для представлення нечітких LR чисел) та lex_solver.py (для вирішеня задачі безпосередньо лексикографічним методом). Модуль lex_solver.py використовує бібліотеки numpy, модулі lr_number.py та simplex_solver.py.

Задамо вхідні параметри та запустимо скрипт:

```
>> C = np.array([LRNumber(2,1,1), LRNumber(3,1,1)])
>> A = np.array([[LRNumber(1,1,1), LRNumber(2,1,1)],
        [LRNumber(2,1,1), LRNumber(1,1,1)]])
>> B = np.array([LRNumber(10,8,14), LRNumber(8,7,13)])
>> solver = FFLPSolver(C, A, B)
>> solver.find_solution("max")
```

Результат виконання:

```
Optimal value: LR number (16.0, 9.000000000000004, 19.0)
X: [(2.0, 0.0, 2.0) (4.0, 1.0, 3.0)]
```

7.4 Тестування модуля задачі нечіткого багаторівневого програмування

В даній роботі було розглянуто модель багаторівневого програмування. Оскільки генетичний алгоритм вимагай досить високої продуктивності, бібліотеку для багаторівневого програмування було розроблено на мові C++.

На рисунку 7.4 наведено результат виконання задачі нечіткого програмування для двох рівнів з використанням генетичного алгоритму. Лідер має вектор рішення $(x_1, x_2\, x_3, x_4,)$, а чотири послідовники мають вектори рішень (y_{i1}, y_{i2}), $i = 1, 2, 3$.

```
Iteration 100
Leader x=(0.594790,8.108600,1.290216), objective F=19.154005.
Follower1 y1=(0.296756,0.298034), f1=0.1743.
Follower2 y2=(6.774973,1.333626), f2=7.2151.
Follower3 y3=(0.646342,0.643853), f3=0.7758.
Iteration 200
Leader x=(0.513420,8.066008,1.343812), objective F=19.217889.
Follower1 y1=(0.256618,0.256802), f1=0.1304.
Follower2 y2=(6.744146,1.321861), f2=7.1242.
Follower3 y3=(0.682953,0.660857), f3=0.8363.
Iteration 300
Leader x=(0.205564,8.062475,1.720873), objective F=20.052807.
Follower1 y1=(0.104189,0.101375), f1=0.0211.
Follower2 y2=(1.316650,6.745824), f2=7.1168.
Follower3 y3=(0.879750,0.841113), f3=1.3039.
Iteration 400
Leader x=(0.144493,8.056235,1.766885), objective F=20.233063.
Follower1 y1=(0.071175,0.073316), f1=0.0104.
Follower2 y2=(6.736789,1.319437), f2=7.1033.
Follower3 y3=(0.878885,0.887999), f3=1.3656.
Iteration 500
Leader x=(0.138224,8.055827,1.773897), objective F=20.337181.
Follower1 y1=(0.069928,0.068295), f1=0.0095.
Follower2 y2=(6.728201,1.327625), f2=7.1018.
Follower3 y3=(0.893593,0.880304), f3=1.3749.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 7.4 – Тестування задачі нечіткого багаторівневого програмування з використанням генетичного алгоритму

7.5 Тестування модуля, що моделює структуру хмари

Для моделювання структури хмари та процесу обслуговування бізнес процесів та універсальних сервісів було створено пакет cloud_solver.py. На рисунку 7.5 наведено програмний код для створення моделі хмари з трьома датацентрами(у Києві, Лондоні, Парижі), де у кожного датацентру по 20 стійок, а кожній стійці по 10 фізичних серверів. Також тут емулюється процес додання віртуальних машин на фізичні сервери, бізнес-процесів та віртуальних сервісів на віртуальні машини. На рисунку 7.6 наведено матриці що показують, скільки ресурсів потребує кожен бізнес-процес чи універсальний сервіс не певній віртуальній машині.

lexicography_lp.py | general_lp.py | cloud_solver.py | crisp_lp.py | flat_number.py | lr_numb

```
if __name__ == '__main__':

    cloud = Cloud("ACTS CLOUD")

    vm1 = VirtualMachine("1", np.transpose(np.array([[1, 2, 3, 4, 5]])))
    vm2 = VirtualMachine("2", np.transpose(np.array([[1, 5, 3, 4, 5]])))
    vm3 = VirtualMachine("3", np.transpose(np.array([[1, 2, 3, 4, 5]])))
    vm4 = VirtualMachine("4", np.transpose(np.array([[1, 2, 3, 4, 5]])))
    vm5 = VirtualMachine("5", np.transpose(np.array([[3, 2, 5, 4, 5]])))

    bp1 = BusinessProcess("A", np.array([1, 2]), np.array([1, 2, 3, 4, 5]))
    bp2 = BusinessProcess("B", np.array([3, 2]), np.array([6, 5, 4, 7, 2]))
    bp3 = BusinessProcess("C", np.array([2, 2]), np.array([3, 2, 6, 4, 1]))

    us1 = UniversalService("A1", np.array([1, 2]), np.array([1, 2, 3, 4, 5]))
    us2 = UniversalService("A2", np.array([1, 2]), np.array([1, 2, 3, 4, 5]))
    us3 = UniversalService("A3", np.array([1, 2]), np.array([1, 2, 3, 4, 5]))
    us4 = UniversalService("A4", np.array([1, 2]), np.array([1, 2, 3, 4, 5]))
    us5 = UniversalService("A5", np.array([1, 2]), np.array([1, 2, 3, 4, 5]))

    cloud.add_VM(vm1)
    cloud.add_VM(vm2)
    cloud.add_VM(vm3)
    cloud.add_VM(vm4)
    cloud.add_VM(vm5)

    cloud.add_BP(bp1)
    cloud.add_BP(bp2)
    cloud.add_BP(bp3)

    cloud.add_US(us1)
    cloud.add_US(us2)
    cloud.add_US(us3)
    cloud.add_US(us4)
    cloud.add_US(us5)

    Cloud.create_byTemplate(["KIEV", "LONDON", "PARIS"], 20, 10)
```

Рисунок 7.5 – Програмний код для створення моделі хмари

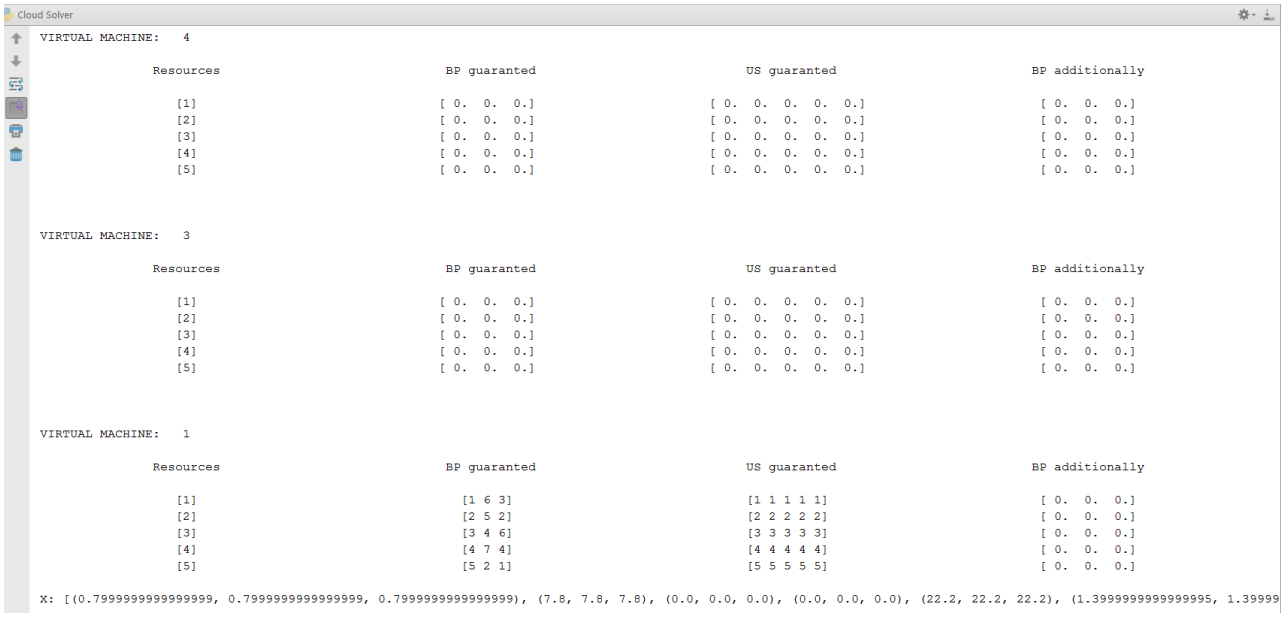

Рисунок 7.6 – Тестування процесу додання бізнес процесів та універсальних сервісів до віртуальних машин

7.6 Висновки про працездатність бібліотеки

Після того як було розроблено модель розподілу ресурсів критичної IT-інфраструктури було реалізовано програмну бібліотеку. Дана бібліотека має декілька модулів. В кожному модулі реалізований певний функціонал, що дає змогу звести складну задачу з розподілу ресурсів до більш простих задач. Для перевірки правильності виконання реалізованих алгоритмів було розроблено набір тестів. Як показали результати, всі модулі даної бібліотеки пройши тести, і результати задовольняють очікуванням.

ВИСНОВОК

У дипломному проекті було розглянуто проблему розподілу ресурсів критичної IT-інфраструктури.

Після детального вивчення проблеми розподілу ресурсів та розгляду предметної області, до вона могла б зустрічатися, було проведено порівняльну характеристику існуючих провайдерів хмарних обчислень, а також моделей розподілу ресурсів критичної IT-інфраструктури. Виявилося, що на даний час не існує такої моделі розподілу ресурсів, яка б задовольняла поставленим вимогам. Були знайдені лише часткові рішення для задач розподілу ресурсів простих систем. Після цього було зроблено декомпозицію системи розподілу ресурсів на підсистеми з метою подальшої їх реалізації. Далі було розглянуто математичний апарат для реалізації нечіткого програмування, а саме: теорію нечітких чисел, заду лінійного програмування на звичайних числах, задачу нечіткого лінійного програмування з лексикографічним методом вирішення, задачу нечіткого багатокритеріального програмування, задачу нечіткого багаторівневого програмування, а також математичну модель для реалізації задачі розподілу ресурсів за трьома рівнями і двома цільовими функціями (надійність та забезпеченість). Потім були розроблені діаграма класів та діаграма варіантів використання для подальшого написання бібліотеки мовами Python та C++. Нарешті, було проведено тестування програмного забезпечення.

В результаті виконання дипломного проекту було розроблено програмну бібліотеку з необхідними структурами даних, моделями та методами, необхідними для моделювання структури хмари а також процесу обслуговування критичних бізнес процесів та універсальних процесів. В подальшому є можливість покращити характеристики процесу розподілу ресурсів за допомогою параметрів генетичного алгоритму, зміни коефіцієнтів та вагових функцій при різних цільових функціях критеріїв оптимізації.

Варто також вказати певні напрямки розвитку даної бібліотеки. Задача, реалізована в даному дипломному проекті є ще досить простою. Проте для

повноцінного використання даної бібліотеки у реальних системах управління критичної IT-інфраструктурою потрібно реалізувати ще декілька складних підсистем. Насамперед – це підсистема сихнронізації підпроцесів задачі розподілу ресурсів критичної IT-інфраструктури. Оскільки вся система має в себе включати такі фунції як моніторинг справності обладнання, видалення непотрібних реплік бізнес-процесів та універсальних сервісів, балансування навантажень, необхідно забезпечити почерговість виконання даних функцій для збереження цілісності інформації щодо розміщення реплік та адекватної поведінки самої системи. Реалізація даних підходів дасть змогу використовувати розроблену програмну бібліотеку для надійного і ефективного управління ресурсами повномасштабних критичних IT-інфраструктур, що в свою чергу призведе до зменшення собівартості обчислювальних ресурсів та забезпечення інформаційними послугами максимальної кількості клієнтів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Дорогий Я. Ю. Критична інфраструктура: вразливості, загрози, ризики / Я.Ю.Дорогий, В.В.Мохор, І.О.Козлюк, В.В.Цуркан // Тези доповідей. ІІ міжнародна науково-практична конференція «Інформаційні технології та взаємодії», 3-5 листопада. – Київ, 2015. – с. 46-47.
2. COBIT 5.0. Российское издание. ISACA. — М.: — 2012. — 94 с.
3. Verissimo P. The CRUTIAL Architecture for Critical Information Infrastructures / P.Verissimo and etc. // Project: IST-FP6-STREP 027513 (CRUTIAL). – pp. 27.
4. Incident handling during attack on Critical Information Infrastructure // Handbook, Document for teachers. – Enisa, 2014. – pp. 24.
5. Tsegaye T. Controls for protecting critical information infrastructure from cyberattacks / T.Tsegaye, S.Flowerday // World Congress on Internet Security (WorldCIS). – London, 8-10 Dec., 2014.
6. Implementing NIST Cybersecurity Framework Using COBIT 5.0. – ISACA, 2014.
7. Теленик С.Ф. Системи управління хмарними IT-структурами / С.Ф. Теленик, О.І. Ролік, М.В. Ясочка, О.С. Квітко // Інтелектуальні системи прийняття рішень і проблеми обчислюваного інтелекту: Матеріали між нар. наук. конф. (16–20 травня) 2011 р. м. Євпаторія. – Том 1. – Херсон: ХНТУ, 2011. – С. 124–127.
8. Теленик С.Ф. Моделі і методи розподілу ресурсів в системах з серверною віртуалізацією / С.Ф. Теленик, О.І. Ролік, М.М. Букасов, О.А. Косован, О.І. Кобец // Зб. наук. праць ВІТІ НТУУ «КПІ». – Випуск № 3. – Київ: ВІТІ НТУУ «КПІ», 2009. – С. 100–109.
9. Кобець О.І. Моделі і методи розподілу ресурсів в системах, побудованих на хмарних обчисленнях [Електронний ресурс]. – Режим доступу: http://intkonf.org/kobets-o-i-modeli-i-metodi-rozpodilu-resursiv-v-sistemah-pobudovanih-na-hmarnih-obchislennyah/.

10. Forrester Research: в 2020 г. рынок публичных облачных вычислений достигнет $241 млрд [Електронний ресурс]. – Режим доступу: http://www.cnews.ru/news/line/forrester_research_v_2020_g.rynok.
11. Aldiab M. Public Cloud War: AWS vs Azure vs Google [Електронний ресурс] / Motasem Aldiab. – 2015. – Режим доступу до ресурсу: https://cloudacademy.com/blog/public-cloud-war-aws-vs-azure-vs-google/.
12. Інформаційна технологія управління проектами/ О. Огірко, Н. Крап-Спісак. – 2016. – С. 59, 60.
13. Моделі управління віртуальними машинами при серверній віртуалізації / С. Ф.Теленик, О. І. Ролік, М. М. Букасов, А. Ю. Лабунський. – 2009. – С. 2–4.
14. Дорогий Я. Ю. Життєвий цикл критичної IT-інфраструктури / Я. Ю. Дорогий. // 5. – 2015. – №2.
15. Didier D. Fuzzy Sets and Systems: Theory and Applications / D. Didier, P. Henri. – New York: ACADEMIC PRESS, 1980. – 393 с.
16. Liu B. Theory and Practice of Uncertain Programming / B.Liu. – UTLAB. – 2009. http://orsc.edu.cn/liu/up.pdf.
17. Edalatpanah A. A New Approach for Solving Fully Fuzzy Linear Programming by Using the Lexicography Method [Електронний ресурс] / A. Edalatpanah, A. Hosseinzadeh. – 2016. – Режим доступу до ресурсу: https://www.hindawi.com/journals/afs/2016/1538496/.